ANTONIO GIULIANA

QUICK BASIC

Tecniche Avanzate di Programmazione

GENNAIO 1992

Antonio Giuliana

QUICK
BASIC

Tecniche Avanzate
di Programmazione

A mia moglie Antonella,
ed al nostro piccolo, Gianmarco

Prefazione

Questo libro è stato scritto nei mesi finali del 1991 e completato nel gennaio 1992. Questa prefazione non esisteva, è stata aggiunta solo adesso per chiarire alcuni motivi per cui questo testo è stato proposto al giorno d'oggi.

Il contenuto del testo, per i tempi, era abbastanza avanzato trattando anche di codice scritto in altri linguaggi (C, Assembly x86) e collegato al QB.

I numerosi esempi di realizzazione di programmi, compreso un piccolo ma elaborato sistema di gestione "finestre in modo testo" tramite una libreria (BPLUS) di cui sono presenti tutti i sorgenti, faceva di questo testo un ottimo strumento per chi aveva scelto il QB come strumento di programmazione per il proprio lavoro ed hobby. Non molti testi, allora, trattavano questi argomenti e sicuramente pochissimi in italiano.

Oggi tutto è cambiato nell'informatica e parlare di sistemi come MS-DOS fa spuntare qualche sorriso. Tuttavia ho notato che ci sono molti giovani che si avvicinano al mondo della programmazione e passano, per vari motivi, anche dal 'vecchio' Quick Basic, senza considerare gli "affezionati" che hanno un po' più anni e che coltivano ancora la passione per il QB anche tramite siti specializzati. Magari lo fanno tramite "emulatori" di MS-DOS e a volte con la "benedizione" di alcuni docenti universitari che intendono fornire delle "basi storiche" a chi inizia.

Ovviamente troverete una terminologia "strana" (che lo lasciato, in molti casi, intatta) e avrete una sensazione inconsueta nel sentire parlare di "floppy disk" o di un uso "opzionale" del mouse, ma la semplice possibilità di "pubblicare" su internet il testo, in un formato fruibile da tanti, mi spinge a *non disperdere il lavoro* fatto tanti anni fa e renderlo comunque disponibile ad un costo simbolico.

Buona lettura.

CAPITOLO 1

Caratteristiche generali

CAPITOLO 1

1.1 CARATTERISTICHE GENERALI

1.1.1 Caratteristiche del sistema

La versione del Quick Basic 4.0 richiede una configurazione di sistema minima di questo tipo:

- un PC-IBM o compatibile con MS-DOS versione 2.1 o seguenti
- almeno un drive per floppy-disk da 360 K
- almeno 320 K di memoria RAM
- qualsiasi scheda video

Comunque è consigliata la seguente configurazione per garantire un lavoro veloce ed efficiente:

- un PC-IBM o compatibile con MS-DOS versione 3.30
- un HD da 20 M ed un FDD da 360 K
- 640 K di memoria RAM
- qualsiasi scheda video

La configurazione del sistema su cui sono stati elaborati i programmi di cui alla libreria BPLUS, era la seguente:

- un AT compatibile con clock a 12 MHz con MS-DOS versione 3.30
- 2 HD da 20 M, 1 FDD da 1.2 M ed 1 FDD da 1.4 M
- 1024 K di memoria RAM di cui 384 usati come disco virtuale
- scheda video tipo VGA

1.1.2 Installazione su floppy-disk

L'installazione del programma su floppy-disk, varia in dipendenza del tipo e del numero dei drive posseduti. Vengono quindi presentate alcune installazioni su dei sistemi con configurazioni diverse di drive per floppy-disk.

a - Un drive per FD da 360 K, 5 pollici e 1/4

In questo caso è necessario preparare 3 dischi, da inserire al momento del bisogno nell'unico drive, i cui contenuti siano i seguenti:

Disco 1

QB.EXE
QB.HLP
(I vostri programmi sorgenti)

Disco 2

BC.EXE

LINK.EXE
LIB.EXE
BRUN40.EXE
BRUN40.LIB

Disco 3

BCOM40.EXE

b - Due drive per FD da 360 K, 5 pollici e 1/4

In questo caso si dovranno preparare i seguenti 4 dischi:

Disco 1

QB.EXE
QB.HLP
MOUSE.COM

Disco 2

BRUN40.EXE
BRUN40.LIB
BQLB40.LIB
LINK.EXE
LIB.EXE
BC.EXE

Disco 3

BCOM40.EXE
QB.LIB

Disco 4

(I vostri programme sorgenti)
QB.QLB
(Altre Quick Libraries)

c - Due drive per FD da 720 K, 3 pollici e 1/2

L'organizzazione dei dischi per questa configurazione, è la seguente:

Disco 1

QB.EXE
QB.HLP
MOUSE.COM
BRUN40.EXE
BRUN40.LIB

Disco 2

BCOM40.LIB
BC.EXE

LINK.EXE
LIB.EXE

Disco 3

(I vostri programme sorgenti)
QB.QLB
(Altre Quick Libraries)

Per quanto riguarda la configurazione con un solo drive per FD da 720 K, 3 pollici e 1/2, la preparazione dei dischi è uguale al caso c. Quando, durante l'uso di Quick Basic, il sistema si blocca emettendo un segnale di 'File non trovato' (File not found), si immette il dischetto corretto nell'unico drive e si digita B: per indicare che il disco con i file richiesti si trova **nell'unità logica B.**

1.1.3 Installazione su hard-disk

Per quanto riguarda l'hard-disk, esistono due diversi modi per effettuare l'installazione. La prima delle due, la più semplice, consiste nel copiare il contenuto dei 3 dischi originali all'interno di una directory del disco fisso.

L'esatta procedura è la seguente:

```
MD C:\QB40          Crea la sottodirectory QB40
     Si immette nel drive A: il disco 1
COPY A:*.* C:\QB40       Copia i file del disco 1
     Si immette nel drive A: il disco 2
COPY A:*.* C:\QB40       Copia i file del disco 2
     Si immette nel drive A: il disco 3
COPY A:*.* C:\QB40       Copia i file del disco 3
```

L'altro modo consiste nel riservare una directory distinta alle librerie (tutti i file con estensione .LIB) e preparare poi le 'variabili d'ambientÈ del DOS.

Più precisamente, dopo aver copiato tutti i file che non hanno estensione .LIB all'interno della directory \QB40 e tutti i file che hanno tale estensione all'interno della directory \QB40\LIB, si predispongono le seguenti variabili di ambiente:

```
PATH=C:\QB40
SET LIB=C:\QB40\LIB
```

1.1.4 Se si dispone di un coprocessore aritmetico

Nel caso in cui si fosse in possesso di un coprocessore aritmetico in virgola mobile, tipo 8087 / 287 / 387, automaticamente questo sarebbe usato dal Quick Basic. In caso contrario, sarebbero usate delle routines di emulazione del coprocessore che assicurano comunque, una precisione elevata nei calcoli.

Se si volesse fare a meno del coprocessore, pur avendolo installato, è sufficiente preparare la variabile di ambiente del DOS chiamata NO87, nel seguente modo:
SET NO87=Uso del coprocessore soppresso

In questo caso la frase presente dopo il segno di uguaglianza viene presentata ogniqualvolta viene eseguito un programma che usa il coprocessore aritmetico. Se non si volesse alcuna frase, basta porre uno o più spazi dopo il segno uguale, e cioè:

SET NO87=___ uno o più spazi ___

Per annullare l'emulazione forzata del coprocessore e tornare ad usare quello installato, si deve preparare la variabile di ambiente suddetta in maniera che sia vuota:

SET NO87=

1.1.5 Se si dispone di un mouse

Quick Basic è predisposto per funzionare con il mouse, anzi è questa la maniera migliore di lavorare con Quick Basic, ma la tastiera affianca tale strumento o lo può sostituire completamente.

Quick Basic lavora con il mouse Microsoft o con un qualsiasi mouse che emuli esattamente il mouse Microsoft. È meglio accertarsi di tale compatibilità tramite il costruttore del vostro mouse.
Per usare il mouse è sufficiente eseguire il programma contenuto nel file MOUSE.COM (versione 6.11 o seguenti) fornito con il Quick Basic, che predispone le routines corrette per la gestione del mouse stesso. È anche possibile installare, se si possiede, il device driver MOUSE.SYS (versione 6.00 o seguenti) al posto di MOUSE.COM ottenendo gli stessi risultati.

1.1.6 Caratteristiche generali di Quick Basic

Il BASIC interpretato, usato fino ad ora da molti programmatori, ha dei vantaggi che, indubbiamente, ne hanno favorito la diffusione. Dopo avere scritto il programma, tramite l'uso di un editor incluso nello stesso interprete, questo può essere eseguito immediatamente senza altri passaggi che rendano il codice comprensibile al sistema. È altresì nota la semplicità con cui è possibile effettuare la correzione dei programmi errati, e la relativa velocità con cui è possibile tornare ad eseguire gli stessi. Tutte queste caratteristiche, indubbiamente vantaggiose, unite alla intrinseca semplicità del linguaggio, hanno fatto del BASIC interpretato il linguaggio di programmazione più usato dai programmatori.

Sono però noti gli svantaggi legati a tale modo di operare ed il più grande, la lentezza di esecuzione dei programmi, è stato risolto tramite l'adozione dei compilatori. Tramite questi ultimi è possibile velocizzare l'esecuzione del programma in maniera tale, molte volte, da renderne possibile l'utilizzo pratico.

È altresì garantita una certa sicurezza che non avvengano manomissioni del testo originale di un programma, in quanto è estremamente difficoltoso, anche se non impossibile, risalire a quest'ultimo attraverso il contenuto del file generato dal compilatore. Non ultimo, esiste anche

il vantaggio fornito da certi compilatori di ampliare il set delle istruzioni e funzioni riconosciute dall'interprete, permettendo di programmare in modo più efficiente e veloce.

Ma lo svantaggio principale che i compilatori comportano , e che ne rende l'uso particolarmente tedioso, è il fatto che, dopo ogni correzione, anche minima, il testo debba essere ricompilato attraverso una sequenza senza dubbio molto lunga.

Molte volte è necessario che un programma sia eseguito solo dopo averlo compilato, sia per motivi di velocità di esecuzione che di collegamento a routines scritte in altri linguaggi, ed il ciclo di correzione è talmente lungo che una sua ripetizione comporta elevati tempi di sviluppo del software. Ecco quindi alcune caratteristiche del Quick Basic che tendono a rendere più conveniente e veloce l'uso del compilatore BASIC:

* Editor interattivo

È la parte principale dell'ambiente Quick Basic. Questo editor permette di inserire le linee del programma, ne controlla immediatamente la sintassi e, se questa è corretta, le traduce in codice eseguibile. Ad alcuni errori di tipo semplice, Quick Basic riesce a rimediare da solo; ad esempio se in un'istruzione di PRINT mancano le virgolette di chiusura di una costante alfanumerica, queste vengono automaticamente aggiunte, come alcuni simboli di interpunzione tra dati (;).

Errori di tipo più grave (errori di sintassi) comportano invece un immediato avviso tramite la descrizione dell'errore stesso. Le parole chiave inoltre, vengono automaticamente convertite in lettere maiuscole se queste erano state scritte in minuscolo.

L'ambiente viene controllato tramite il mouse e la tastiera o solo tramite la tastiera, attraverso dei menu a tendina relativi alle funzioni esplicate. Inoltre, molti dei tasti e delle combinazioni di tasti che controllano l'editing del testo, sono quelli già conosciuti da chi usa Wordstar ©, noto word processor usato in ambiente IBM ©.

* Aiuto immediato

Sono previste molte schermate di aiuto riguardanti i tasti e le combinazioni dei tasti con le funzioni associate, una tavola ASCII disponibile come pro-memoria e delle pagine di aiuto sulla sintassi e funzione di tutti i comandi, istruzioni e funzioni eseguibili dal compilatore.

È possibile richiedere l'aiuto direttamente per quanto riguarda una singola istruzione, attraverso la modalità 'context-sensitivÈ, modalità che fornisce delle pagine di aiuto sull'istruzione su cui il cursore si trova in un determinato momento.

* Esecuzione istantanea del programma sorgente

Il programma appena scritto viene eseguito immediatamente, alla stregua di un interprete, e non viene compilato separatamente dato che ogni istruzione è compilata e controllata all'atto dell'immissione.

La velocità di esecuzione è quella del programma compilato. La compilazione avviene nella memoria centrale del computer e, dopo aver corretto eventualmente il programma, questo è pronto per essere rieseguito immediatamente senza altri passi intermedi.

* Correzione interattiva dei programmi sorgenti

L'esecuzione di un programma può essere interrotta in qualsiasi momento, ed il contenuto delle variabili può essere testato in maniera diretta (cosa impossibile da farsi con un normale compilatore). La stessa può essere ripresa, alla maniera di un interprete, dal punto in cui era stata interrotta; l'esecuzione può avvenire inoltre, passo-passo, cioè istruzione dopo istruzione.
È possibile testare continuamente il contenuto di ogni variabile e risulta molto comodo l'uso di una pagina video dedicata all'editor separata dalla pagina dedicata al programma utente. Esistono i breakpoints, e cioè dei punti in cui è previsto, per motivi di correzione, che l'esecuzione del programma si arresti per permettere il test delle variabili.
È disponibile inoltre, il Code View ©, programma tramite il quale è possibile eseguire e correggere direttamente il file .EXE ottenuto dopo la definitiva compilazione del programma sorgente tramite il compilatore BC.

* Creazione veloce di un file eseguibile

La creazione di un file eseguibile è possibile tramite il compilatore di linea BC. Quando si è soddisfatti di come un programma gira in ambiente Quick Basic, questo può essere trasformato in un file di tipo .EXE eseguibile in DOS.

È possibile effettuare questa trasformazione sia in ambiente Quick Basic che in DOS (tramite il compilatore BC).

* Finestre di editing multiple

Si possono usare più finestre video su cui far apparire il testo del programma sorgente in uso, per vederne diverse parti contemporaneamente e permetterne una più efficiente correzione.

* Tutti i modi grafici sono supportati

Tramite il Quick Basic è possibile supportare tutti i modi grafici possibili; le schede MGA, CGA, MCGA, EGA, VGA sono supportate direttamente e la scheda Hercules © è supportata tramite l'uso di un programma residente fornito con il Quick Basic (QBHERC.COM).

* Moduli multipli in memoria

Esiste la possibilità di caricare in memoria diversi programmi da editare ed eseguire ed è possibile creare le versioni .EXE dei programmi automaticamente con un solo comando.

* Generazione e gestione automatica delle librerie

La generazione delle librerie è automatica in Quick Basic, e le librerie di tipo .QLB (Quick Libraries) necessarie al funzionamento dei programmi in ambiente Quick Basic, sono create attraverso le librerie di tipo .LIB.
In ogni momento è possibile modificare le librerie .QLB apportando le stesse modifiche effettuate in quelle .LIB, automaticamente e velocemente.

1.1.7 Differenze con le precedenti versioni

- Differenze con la versione 3

Rispetto alla versione 3, le seguenti istruzioni e funzioni sono state aggiunte o migliorate:

AS, CALL, CLEAR, CLNG, COLOR, CVL, DECLARE, DEFLNG, DIM, FILEATTR, FREEFILE, FUNCTION, GET, LCASE$, LEN, LSET, LTRIM$, MKL$, OPEN, PALETTE, PUT, RTRIM$, SCREEN, SEEK, SETMEM, STATIC, TYPE, UCASE$, VARPTR, VARSEG.

- Differenze con la versione 2

Rispetto alla versione 2, oltre quanto detto per le differenze con la versione 3, esistono altre funzioni ed istruzioni che sono state aggiunte o migliorate, cioè:

CASE, CLS, CONST, CVSMBF, CVDMBF, DO…LOOP, EXIT, MKSMBF$, MKDMBF$, SELECT CASE, WIDTH.

Rispetto alle versioni precedenti inoltre, la versione 4 è compatibile per quanto riguarda i file sorgenti ma non sono compatibili i file oggetto. È quindi necessario ricompilare i sorgenti con il Quick Basic versione 4 per poterli eseguire.

1.1.8 Importanti caratteristiche aggiunte al linguaggio

* TYPE..END TYPE

Come per le strutture in C e per i records in PASCAL, è possibile definire dei nuovi tipi di dati attraverso una unione di dati di tipo primitivo. Le strutture TYPE..END TYPE semplificano l'I/O dei file di tipo random.

* Interi lunghi (32 bits)

Sono adesso disponibili i dati interi lunghi (32 bits) con cui è possibile trattare dati numerici interi compresi nell'intervallo -2147483648…+2147483647. Il trattamento di tali dati è più accurato rispetto a quello dei valori in singola precisione corrispondenti in quanto esenti da errori di arrotondamento. Sono state aggiunte le funzioni di conversione per il trattamento di tali dati con i file random in cui occupano 4 bytes.

Il simbolo caratteristico per la definizione di dati di tale tipo è l'ampersand (&).

* Ricorsione

È possibile che le funzioni o i sottoprogrammi siano ricorsivi, cioè che chiamino sé stessi un certo numero di volte, come in altri linguaggi come C o PASCAL.

* Accesso binario per i file

L'accesso binario, unito al sequenziale ed al random per i file, permette adesso la lettura e la modifica dei file di tipo non ASCII.

* Formato numerico IEEE

Si può usare con il Quick Basic 4.0 anche il formato dei numeri IEEE per garantire una migliore precisione nel calcolo floating-point.

1.2 IL COMANDO QB

1.2.1 Il comando QB ed i relativi parametri

Il comando QB dato sotto DOS permette il caricamento e l'esecuzione del Quick Basic che si presenta con la schermata caratteristica del suo ambiente.

Tuttavia, dopo il comando QB è possibile aggiungere una serie di parametri per cui il formato completo del comando prende la seguente forma:

QB [[/RUN][file sorgente][/AH][/B][/C:grandezza buffer][/G][/H]
[/L[nome libreria]][/MBF][/CMD stringa]]

La descrizione dei parametri suddetti è la seguente:

file sorgente È il nome del file che viene automaticamente caricato in memoria al momento della partenza. È inteso che, se l'estensione non è specificata, il sistema aggiunge automaticamente .BAS. Se si vuole specificare un nome di file che non ha estensione, bisogna aggiungere il punto dopo il nome stesso senza che sia seguito da altro. Ad esempio, se si vuole caricare in memoria il file di nome PROVA, bisogna che il comando sia:

QB PROVA.

/RUN file sorgente Il file specificato attraverso il nome, oltre ad essere caricato in memoria viene automaticamente eseguito.

Ad esempio, per eseguire il file PROC.BAS il comando è:

QB /RUNPROC

/AH Permette che gli arrays dinamici di numeri, di records e di stringhe di lunghezza fissa, possano occupare più di 64 K di memoria. Infatti, normalmente essi non possono oltrepassare tale limite.

/B Permette di usare Quick Basic in bianco e nero se si dispone di un monitor a colori o se si dispone di una scheda che emula il colore sui monitors monocromatici con le tonalità di grigio. Quick Basic è capace infatti di determinare il tipo di scheda grafica ma non il tipo di monitor del sistema.

/G In molti sistemi che hanno la scheda grafica CGA, permette di aggiornare più velocemente lo schermo. Se si vedono dei disturbi quando si aggiornano larghe aree dello schermo (per esempio durante lo scorrimento del testo) allora il sistema non gestisce l'aggiornamento rapido dello schermo e non è consigliato quindi l'uso di tale parametro.

/H Abilita il funzionamento di Quick Basic con il formato dello schermo più grande che il sistema permette. Con la scheda EGA o VGA vengono visualizzate 43 linee di 80 colonne.

/MBF Con tale parametro, tutte le funzioni di conversione numerica lavorano nel Microsoft Binary Format (MBF) e non nel formato IEEE.

/C:grandezza buffer Predispone la grandezza del buffer di ricezione per le comunicazioni seriali. Il valore di default è 512 ma questo può essere variato fino a 32767. Il buffer di trasmissione ha invece la lunghezza di 128 byte che non è modificabile in alcun modo. Tutto ciò ha effetto solo se è presente almeno una scheda di interfaccia seriale.

/L nome libreria Carica in memoria la libreria di tipo .QLB specificata. Se non ne viene specificata nessuna, è caricata la libreria standard QB.QLB. Ad esempio per caricare la libreria BPLUS.QLB, è sufficiente immettere il comando;

QB /LBPLUS

/CMD stringa Questo parametro, che deve essere l'ultimo nella linea di comando, permette di passare dei parametri, sotto forma di stringa, alla funzione COMMAND$ di Quick Basic.
È possibile modificare il valore ritornato dalla funzione COMMAND$ tramite il menu a tendina **Run** (**Esegui**) di Quick Basic.

1.2.2 La configurazione dello schermo di Quick Basic

Si può personalizzare lo schermo di Quick Basic eseguendo la scelta **Options** nel menu **View** (scelta **Display** nel menu **Opzioni**). Viene presentata una schermata (dialog box) in cui, tramite il tasto di tabulazione, è possibile scegliere il colore del testo e di sfondo del testo normale, dell'istruzione corrente e delle linee di breakpoint e puntualizzare degli attributi (lampeggio, alta luminosità) di tale testo.

È inoltre possibile non visualizzare le barre di scorrimento (Scroll Bars) ed è possibile definire il numero di spazi che vengono visualizzati tramite il tasto di tabulazione.

I colori possono essere cambiati tramite i tasti di posizionamento del cursore in alto o in basso e gli attributi tramite lo spazio. Il tasto RETURN conferma tutti i dati forniti e salva la configurazione dello schermo nel file QB.INI.

1.2.3 Il file QB.INI

Nel file QB.INI viene quindi salvata la configurazione dello schermo di Quick Basic automaticamente all'uscita dalla precedente situazione.

Le informazioni salvate in questo file verranno automaticamente recuperate all'apertura di una nuova sessione di lavoro in Quick Basic. Per ripristinare i colori di default, basta cancellare, tramite il DOS, il file QB.INI, in quanto, in assenza di tale file, vengono assunti i valori iniziali.

I valori dei parametri che è possibile variare, sono contenuti all'interno del file QB.INI secondo il seguente schema:

Offset dall'inizio del file	Parametro
04h	Colore testo normale background
06h	Colore testo normale foreground
08h	Alta luminosità testo normale
0Ah	(Lampeggio testo normale) **(*)**
0Ch	Colore istruz. corrente background
0Eh	Colore istruz. corrente foreground
10h	Alta luminosità istruz. corr.
12h	Lampeggio/Sottol. istruz. corr.
14h	Colore linee di breakp. backgr.
16h	Colore linee di breakp. foregr.
18h	Alta luminosità linee di breakp.
1Ah	Lampeggio/Sottol. linee di breakp.
1Ch	Numero di spazi per il tab
1Eh	Visualizzazione scroll bars

(*) Il lampeggio del testo normale non può essere attivato

Le posizioni e i contenuti del file QB.INI differiscono dalla versione 4.0 alla 4.5.

CAPITOLO 2

L’ambiente di sviluppo dei programmi

CAPITOLO 2

2.1 L'AMBIENTE DI SVILUPPO DEI PROGRAMMI

2.1.1 Lo schermo di Quick Basic

All'apertura di una sessione di lavoro con il Quick Basic, tramite il comando QB (v. 1.2.1), appare la schermata dell'ambiente di lavoro. Questa appare divisa in due parti fondamentali, la finestra in cui sarà visualizzato il testo dei vostri programmi in alto, e in basso, una finestra, di dimensioni più ridotte, in cui potranno essere eseguite immediatamente le istruzioni del linguaggio; la finestra in alto è la **View Window** e, lo vedremo in seguito, potrà essere divisa in 2 finestre più piccole, mentre la finestra in basso è la **Immediate Window**.

In alto è visualizzata una linea dove compare il nome di ogni menu con cui si potrà lavorare e, all'estrema destra, l'indicazione che il tasto F1 servirà ad ottenere aiuti sia a livello generale che sulla singola istruzione; questa linea è chiamata **Menu bar**.

In basso, all'ultima linea, compare l'indicazione di copyright del Quick Basic ed il numero della versione, ma questa indicazione è temporanea in quanto quest'ultima linea servirà a visualizzare alcune informazioni sullo stato del compilatore; infatti, se si prova a premere il tasto del controllo del cursore che lo porta a destra di una posizione, si può notare la scomparsa delle precedenti informazioni.
Al posto di queste sarà indicato il nome del modulo attivo (**Main**) che, all'inizio, sarà <Untitled> (cioè senza titolo) in quanto non abbiamo ancora indicato il nome del nostro programma. Al centro della linea appare l'indicazione **Context** che è seguita dal nome del modulo e della procedura, se è il caso, che contiene la prossima istruzione da eseguire; nel caso in cui, come all'apertura, non sia in esecuzione nessun modulo, l'indicazione precisa appunto che nessun programma è in esecuzione (Program not running).
A destra viene indicato lo stato del tasto Caps Lock, che controlla le maiuscole, e del tasto Num Lock, che controlla il tastierino numerico; se il primo è attivo viene evidenziata una C e se lo è il secondo, viene indicata una N.
Per ultima, a destra, viene visualizzata l'indicazione della posizione del cursore all'interno del testo; il primo numero indica il numero della linea mentre il secondo quello della colonna. Questi valori sono aggiornati, oltre che durante l'editing, anche durante il caricamento di un file sorgente da disco e durante la compilazione di un testo. La suddetta linea dove compaiono tali indicazioni, è denominata **Status bar**; durante la scelta delle opzioni dai menu, viene visualizzata su questa linea, una semplice spiegazione delle funzioni svolte dalla opzione corrente.

A destra ed in basso sono visualizzate inoltre, le **Scroll Bars**, due barre in cui viene indicata la posizione del testo visualizzato nella view window relativamente a tutto il testo del programma in memoria, in senso orizzontale una ed in verticale l'altra. I quadratini che si spostano su queste barre, e che esplicano la funzione suddetta possono essere spostati con il mouse, per permettere lo spostamento all'interno del testo. Tali barre non sono attivate quando ci si trova nella finestra Immediate.

È interessante notare che questa schermata viene presentata senza alterare quella del programma che, eventualmente, fosse in esecuzione; infatti Quick Basic predispone uno

schermo per l'utente, a cui faranno riferimento tutte le istruzioni di visualizzazione, ed uno schermo per sé stesso, che contiene le informazioni suddette. In un qualsiasi momento durante l'editing dei programmi, tramite il tasto F4, si può passare da uno schermo all'altro, alternativamente.

2.1.2 Scelta delle opzioni dai menu

Il Quick Basic mette a disposizione dell'utente 7 menu più quello di aiuto con cui è possibile effettuare tutte le operazioni possibili. Questi menu sono del tipo a tendina e quindi sono visualizzati in verticale solo a richiesta. Per poterlo fare basta premere il tasto **Alt ed il tasto dell'iniziale del nome del menu**. Ad esempio, per visualizzare il menu File, basta premere il tasto Alt e la lettera F; le opzioni relative verranno presentate immediatamente.

Per scegliere tra le varie opzioni esistono due metodi: si ci può spostare usando le frecce di posizionamento del cursore in basso o in alto e premere il tasto Return o, in alternativa, si può premere la lettera evidenziata nella opzione, generalmente l'iniziale, senza la necessità di premere il tasto Return.

Se si dovesse avere sbagliato nella scelta del menu appena aperto, si può rimediare con il tasto Esc per abbandonare l'operazione per poi scegliere il menu corretto o, direttamente, da quello sbagliato, tramite i tasti di posizionamento del cursore a destra o a sinistra, si apre il menu desiderato.

Se si è in possesso di un mouse la scelta dei menu avviene, dopo avere posizionato il cursore sul menu prescelto, con il tasto sinistro del mouse stesso. La medesima operazione va effettuata per scegliere l'opzione all'interno del menu e per abbandonare l'operazione, basta invece muovere il mouse fuori dal menu selezionato e premere il tasto a sinistra.

2.1.3 Uso delle finestre

La schermata con cui si presenta il Quick Basic è divisa in più parti (v. 1.3.1) chiamate finestre (windows).

La finestra dove viene visualizzato il testo del nostro programma (View window) è, inizialmente, quella selezionata. Per potere cambiare questo stato di cose e selezionare la finestra in cui potere eseguire immediatamente le istruzioni Basic (Immediate window), il Quick Basic mette a disposizione il tasto funzione F6. Tramite questo tasto, può essere selezionata la finestra che serve, tenendo presente che la selezione avverrà ad ogni pressione di F6, dalle finestre più in alto verso quelle più in basso. Per ottenere una direzione di selezione contraria, basta premere insieme a F6 il tasto Shift; in questo caso le finestre saranno selezionate in senso contrario.

Una finestra è selezionata quando il titolo di quest'ultima appare evidenziato ed il cursore si posiziona all'interno della stessa.

Con il mouse, basta spostarsi nella finestra prescelta e premere il tasto a sinistra nella stessa.

La finestra View può essere divisa orizzontalmente in due parti, parti che possono visualizzare diversi programmi o diverse parti dello stesso programma, al fine di migliorare gli spostamenti

di testo che si possono rendere necessari per l'editing degli stessi. Per realizzare tale funzione, dopo avere scelto il menu View (con i tasti Alt e V), si seleziona l'opzione Split e si preme Return o si preme la lettera P; immediatamente la finestra View viene divisa in due parti uguali.

La finestra Immediate è necessaria quando si vogliono eseguire immediatamente alcune istruzioni. La linea delle istruzioni separate dal simbolo di separazione (:), può essere lunga al massimo 256 caratteri. Ad esempio, se si volesse cancellare il video basterebbe, dopo avere selezionato con il tasto F6 la finestra Immediate, immettere il comando CLS seguito dal tasto Return; dopo avere eseguito il comando ed avere premuto un qualsiasi tasto, il controllo torna all'ambiente Quick Basic. Bisogna notare che lo schermo a cui si fa riferimento per la cancellazione è quello dell'utente, come per qualsiasi altro comando di visualizzazione (v. 1.3.1).

Le dimensioni di tutte le finestre possono essere cambiate, aumentate o diminuite, per rendere più agevole la correzione dei programmi. Il cambiamento delle dimensioni è effettuato solo sulla finestra attiva ed è effettuato dai seguenti tasti:

Alt +	Espande di una linea la finestra
Alt -	Contrae di una linea la finestra
Ctrl F10	Aumenta le dimensioni della finestra attiva al massimo

Premendo nuovamente questi tasti, le finestre tornano ad avere le dimensioni precedenti.

2.1.4 La dialog box

L'uso della dialog box è molto diffuso in Quick Basic dato che in tutte le occasioni si ha a che fare con questi caratteristici elementi dell'ambiente. Una dialog box è una finestra al cui interno sono visualizzati dei dati o delle opzioni da scegliere in base all'operazione che si vuole effettuare. Ad esempio, nel caso in cui dal menu File si decida di caricare un file di programma dal disco, scegliendo l'opzione Open Program, viene presentata la dialog box corrispondente in cui viene visualizzato il contenuto della directory corrente del disco di default; inoltre, sono presentate altre informazioni tra le quali ci si può muovere per selezionarle. Si è già visto come muoversi all'interno della dialog box in occasione della personalizzazione dello schermo del Quick Basic (v. 1.2.2).

2.2 IL MENU FILE

2.2.1 I diversi tipi di file

In Quick Basic esistono diversi tipi di file ma, sicuramente, i **programmi** sono i più importanti. Questi sono sequenze di istruzioni caratteristiche del linguaggio e, normalmente, hanno estensione .BAS. Se avete finora programmato con un interprete Basic in cui ad ogni programma corrisponde un file, vi accorgerete che in Quick Basic le cose non stanno proprio in questo modo, anche se per programmi semplici, questa regola vale ancora.

Quando i programmi cominciano ad avere dimensioni molto elevate è conveniente dividerli in parti più piccole e, soprattutto, se questi hanno in comune delle sottoroutines o delle funzioni,

è bene lavorare con i **moduli**. In un modulo può essere contenuta una parte di programma e più moduli caricati insieme in memoria, quando sono tutti parte di un unico programma, facilitano la realizzazione finale dello stesso in quanto si può agire su parti più piccole e ben determinate. È chiaro a questo punto che un programma in Quick Basic corrisponde almeno ad un modulo; ogni modulo può occupare fino a 64 K. Dei moduli se ne tornerà a parlare, in maniera più approfondita, in seguito (v. 1.4.3).

All'interno di un programma scritto in Quick Basic, può esserci un riferimento ad un file del tipo **include**. In questa categoria di file, possono essere scritte tutte quelle istruzioni, generalmente frasi COMMON e dichiarazioni di SUB e FUNCTION (v. 2.2.5), che sono necessarie al funzionamento del programma stesso. Ai fini della compilazione, il contenuto dei file include è parte integrante del file programma, anche se, nel programma stesso, esiste solamente il riferimento al file include tramite il metacomando $INCLUDE. Questi file si usano per evitare l'inserimento di tutte le frasi di dichiarazione delle routines usate da un programma, specialmente quando queste routines fanno parte di una libreria creata dall'utente, come la BPLUS (v. cap. 7); basta mettere la frase di $INCLUDE per avere le dichiarazioni delle routines della libreria già fatte in un solo colpo.

Ultimo tipo di file gestito dal Quick Basic è quello chiamato **documento**. In occasione dell'editing di un file documento, il Quick Basic si comporta come un qualsiasi word processor; non vengono eseguiti i controlli di sintassi sulle parole contenute nel testo ed il file è inteso sempre indipendente da altri (nel caso di un programma multi-modulo viene caricato solo il file prescelto).

Non è possibile caricare lo stesso file più volte sotto una diversa classificazione. Un file cioè, non può essere caricato prima come documento e poi come programma o viceversa. I file già caricati, sono indicati nella dialog box attivata dal tasto F2.

2.2.2 Gestione dei programmi

All'apertura di una sessione di lavoro in un ambiente Quick Basic, si è in modo programma. L'editor del Quick Basic è pronto per immettere, modificare ed eseguire istruzioni caratteristiche del linguaggio. Per potere lavorare con un programma realizzato in precedenza, bisogna scegliere l'opzione **Open Program** (**Apri programma**) del menu File con cui si attiva la dialog box che ci indica con quale drive stiamo lavorando, quale è la directory corrente e ci fornisce la lista dei file di tipo .BAS e le sottodirectory presenti nella directory suddetta. Per caricare un file basta introdurre al posto dell'indicazione *.bas (che d'altronde scompare alla prima battuta), il nome, eventualmente completo di drive e path, e premere Return. Se il file esiste, questo viene caricato mentre tutti quelli eventualmente già presenti vengono eliminati dalla memoria; in caso di errore, il messaggio File not Found viene presentato e niente viene modificato.
Un altro modo di procedere è quello di spostarsi sulla lista dei file che è evidenziata nella dialog box, e scegliere tramite il tasto Return, il file voluto dopo averlo selezionato con i tasti di controllo del cursore (freccia in basso ed in alto); per spostarsi velocemente all'interno della lista si può usare il tasto con l'iniziale del file che ci interessa.

Si possono anche usare i caratteri jolly (* e ?) nel nome del file ottenendo soltanto che la lista visualizzata comprenda tutti i file selezionati, e poi si può scegliere tra questi; alla stessa

maniera, per vedere la lista di un disco diverso dall'attuale, basta indicarne il nome con la directory prescelta.

Se nella finestra non possono essere visualizzati contemporaneamente tutti i file, il contenuto della stessa può essere fatto scorrere a destra e a sinistra per esaminare tutti i nomi, nomi che vengono elencati sempre in ordine alfabetico.

Se si volessero controllare i file contenuti in una directory il cui nome è specificato nella lista sul video, basta selezionare la stessa; ma attenzione che la directory selezionata **non diventa la directory corrente** che sarà sempre quella di partenza; per cambiare directory di lavoro servirsi del comando CHDIR in modo Immediate (v. 1.3.3).

Per caricare un file con il mouse è sufficiente un doppio click sul tasto del mouse stesso, dopo avere posizionato il cursore nella finestra sul nome del file scelto.

Per registrare sul disco il programma esistente in memoria, con tutte le modifiche eventualmente apportate, esistono 3 diverse opzioni nel menu File **Save, Save As…, Save All** (**Salva, Salva con nome…, Salva tutto**).
La prima si usa per registrare il contenuto del modulo corrente mentre l'ultima è usata quando si devono registrare tutti quei file presenti in memoria su cui sono state fatte delle modifiche dall'ultimo salvataggio su disco; questa opzione è molto comoda quando si lavora con diversi moduli e quindi si modificano diversi file. La seconda opzione (Save As…) apre una dialog box con cui si può specificare il nome che il modulo deve avere sul disco e il formato di registrazione prescelto. Infatti il modulo può essere registrato in formato ASCII, come un qualsiasi testo, in modo da renderlo leggibile ad altri programmi, o in formato Quick Basic per una più veloce scrittura/lettura dello stesso; nel secondo caso il testo viene codificato. Il file salvato in quest'ultimo modo si riconosce dal precedente perché inizia sempre con un carattere che ha codice Ascii 253; quando un file viene salvato in modo testo, il primo carattere dello stesso **non può mai assumere** tale valore dato che deve appartenere alla prima parte della tabella (codici Ascii inferiori a 128). L'opzione Save As… è comoda quando si vuole registrare un modulo con un altro nome per evidenziare il fatto che alcune modifiche non sono presenti sul file precedente.

I file di tipo Include e Documento sono **sempre** salvati in formato ASCII. Nel caso in cui si voglia abbandonare un modulo appena modificato, sia per caricarne un altro che per uscire dall'ambiente Quick Basic, viene sempre presentata una dialog box con un messaggio di avvertimento e viene data la possibilità di registrare il modulo stesso prima di abbandonarlo; se il modulo non ha ancora nome (è il caso di un modulo creato ma non salvato) viene visualizzata la dialog box di cui all'opzione Save As… che permette di assegnare un nome corretto al modulo stesso.

Quando si vuole creare un nuovo programma e si deve eliminare dalla memoria uno o più moduli già presenti si deve usare l'opzione **New Program** (**Nuovo programma**) del menu File, che permette la pulizia dell'area di lavoro per fare posto ad un nuovo programma. I moduli precedentemente in memoria, se modificati dopo il loro ultimo salvataggio, possono essere automaticamente salvati sul disco previo avvertimento del Quick Basic. Il programma nuovo non ha, inizialmente, alcun nome (Untitled).

Per fondere il contenuto di un file con quello del programma correntemente visualizzato, l'opzione **Merge** è usata allo scopo. Il file da unire viene prescelto attraverso una dialog box che è del tutto simile a quella usata dall'opzione Open Program, ed il testo da fondere viene inserito dalla posizione che il cursore ha sul video rispetto al programma in memoria. Non è possibile che i due file da fondere insieme abbiano delle SUB o delle FUNCTION con lo stesso nome.

2.2.3 Le SUBs e le FUNCTIONs

I programmatori Basic avranno già una certa confidenza con i concetti di 'sottoroutinÈ e di 'funzione utentÈ, ma un breve richiamo a questi concetti è comunque utile.

All'interno dei programmi, molto spesso è necessario che alcune istruzioni siano eseguite un certo numero di volte in più punti del programma stesso. Al fine di evitare un enorme spreco di spazio e migliorare l'organizzazione del programma, queste istruzioni ripetitive vengono raggruppate e a questo insieme viene dato nome di sottoroutine. Questo insieme di istruzioni verrà eseguito dal programma principale ogni volta che bisognerà tramite una semplice chiamata; il controllo ritornerà, dopo che la sottoroutine sarà stata eseguita tutta, al punto in cui il programma la aveva chiamata così da permettere lo svolgimento della rimanente parte dello stesso.

Come ulteriore vantaggio, la sottoroutine, una volta creata, può essere utilizzata da tutta una serie di programmi; essa diventa quindi uno strumento di programmazione, al pari di una qualsiasi istruzione del linguaggio. È questo il punto centrale che bisogna chiarire per comprendere a fondo l'importanza di quelle che si chiamano **librerie**. Esse infatti, non sono altro che un insieme, una collezione di sottoroutines e permettono di aumentare ulteriormente le potenzialità del linguaggio.

Le sottoroutines, di regola, non ritornano alcun valore, di nessun tipo, al programma che le ha chiamate, ma sono usate per compiere le più diverse azioni, come stampe, visualizzazioni, letture o scritture di file, mentre, se un valore deve essere ritornato in seguito ad una elaborazione che può essere, ad esempio, di tipo numerico, allora si parla di **funzioni utente**. In Quick Basic esistono delle funzioni, delle istruzioni cioè che elaborano i dati e che restituiscono un valore, che sono già definite; abbiamo la funzione che calcola il valore della radice quadrata di un numero, o quella che calcola il suo logaritmo naturale ma sono funzioni di base. Se si volesse una funzione che calcoli la radice quadrata del logaritmo di un numero allora la si dovrebbe definire utilizzando le funzioni di base già disponibili ed in questo caso si parlerebbe di funzione definita dall'utente o di funzione utente.

La gestione delle sottoroutines in Basic è affidato alle istruzioni GOSUB…RETURN mentre le funzioni utente sono definite dalla frase DEF FN…(). Questo modo di procedere esiste anche in Quick Basic, ma è limitativo e per questo il Quick Basic introduce il concetto di SUB e di FUNCTION.

Il testo che viene immesso tra la parola SUB e la parola END SUB è del tutto simile a quello che costituiva la vecchia sottoroutine che si concludeva con l'istruzione RETURN, ma l'uso della SUB comporta i seguenti vantaggi:

* uso di variabili locali

Le variabili usate all'interno della SUB sono, normalmente, di tipo locale; sono cioè create al momento dell'esecuzione della SUB ed il nome che viene dato loro può essere uguale a quello che altre variabili hanno nel programma principale, pur continuando ad essere distinte. Alla fine della SUB tutte le variabili locali sono distrutte in quanto inutili (questa caratteristica richiama il funzionamento del linguaggio C).

* uso nei programmi multi-modulo

Nei programmi multi-modulo (v. 3.1.4) tutti i moduli possono eseguire una SUB in quanto essa è 'visibilÈ da tutti (tutti i moduli cioè, possono richiamare la SUB). Al contrario, una sottoroutine scritta nella vecchia forma, è eseguibile solo dal modulo in cui è scritta.

Per le FUNCTION tutto ciò che si è detto rimane valido, considerando che queste ritornano un valore, che può anche non essere numerico, al programma chiamante. Anche in queste le variabili usate sono locali ed è possibile sfruttarle da diversi moduli, ma esistono altre caratteristiche che le rendono migliori delle funzioni definite con DEF FN…

* ricorsività

Una FUNCTION può richiamare sé stessa e realizzare quindi il concetto di ricorsività.

* complessità

Si possono scrivere funzioni molto complesse, in quanto è possibile scriverle su più linee e sfruttare tutte le istruzioni del linguaggio.

* nessuna restrizione del nome

Le funzioni definite con DEF FN… sono usate sempre con il prefisso FN davanti al nome. Questa restrizione non è valida per le FUNCTION.

Appare evidente quindi, l'importanza di tali strutture nella programmazione in Quick Basic, e per quanto riguarda la loro creazione e gestione, essa è facilitata da diverse opzioni facilmente selezionabili dai menu (v. 2.3.7).

2.2.4 I moduli ed i file .MAK

È meglio usare un esempio che chiarisca l'importanza dei moduli. Il seguente schema dimostrativo di programma per la gestione dei Conti Correnti, è stato creato appositamente allo scopo. Osserviamone la struttura:

CC00
CC01
 CC01
 SHOWMENU
CC02

CC06
 CC06
CCSU
 HERROR
 KWAIT

CC02	LINEONOFF
CC03	LMASK
CC03	MKRECORD
CC04	OGGI
CC04	STATUS

Questa struttura è visualizzata tramite il tasto F2, dopo che tutti i moduli sono stati caricati in memoria. Un modulo è, ad esempio, quello denominato CCSU che ha delle SUB e delle FUNCTION chiamate HERROR, KWAIT, LINEONOFF, LMASK, MKRECORD, OGGI e STATUS; ma è un modulo anche CC01 che ha le sue SUB di nome CC01 e SHOWMENU. Ognuno di questi moduli concorre al funzionamento di tutto il programma che altro non è che l'insieme di tutti questi moduli. Con questa organizzazione, in ogni modulo può esserci un riferimento alle SUB e alle FUNCTION appartenenti agli altri moduli e i dati possono essere scambiati tra tutti i moduli stessi. L'ultimo modulo (CCSU) è il Main; è quello che contiene le istruzioni di partenza ed è quello che richiama tutti gli altri. Un tipo di programmazione simile si realizza con il linguaggio C e con altri linguaggi simili, dove dalla funzione main è possibile eseguire tutte le altre con un meccanismo che ricorda le scatole cinesi (tenere presente che il programma dell'esempio precedente non è incluso in questo testo per motivi di spazio).

Quando un programma è multi-modulo (è composto cioè da molti moduli) come nel caso dimostrativo precedente, viene creato da Quick Basic un file con estensione .MAK che contiene la sequenza dei nomi dei moduli appartenenti al programma. È tramite questo file che tutti i moduli vengono richiamati, automaticamente, in memoria e questo avviene in due casi:

a. Tramite l'opzione Open Program si carica il file che contiene il main-module (nel caso dell'esempio, CC00.BAS).

b. Sempre con l'opzione Open Program si carica il file che ha estensione .MAK e che contiene il progetto del programma; questo ha il nome del file che contiene il main-module (nell'esempio, CC00.MAK).

In ambedue i casi, sono caricati in memoria, in sequenza automatica, tutti i moduli che fanno parte del programma e la loro lista è ottenibile alla fine, usando il tasto F2.

Dato che il file .MAK è un file di tipo ASCII che contiene solamente i nomi dei file che contengono i moduli componenti il programma, è buona norma, quando si intende creare un programma multi-modulo, gestire il contenuto del file in questione, aggiungendo i nomi dei file dopo averlo caricato in modo Documento.

È altresì possibile sfruttare le 3 opzioni **Create File…, Load File…** e **Unload File** (**Crea file, Carica file, Chiudi file**) che servono anche a gestire la creazione di programmi multi-modulo. Usando la prima opzione è possibile creare un nuovo file, di tipo modulo, include o documento, come è indicato dalla dialog box che viene aperta. Viene richiesto inoltre il nome del file e questo viene aggiunto alla lista di quelli eventualmente presenti in memoria; solo usando i comandi di tipo Save o nei casi in cui il salvataggio è automatico, il contenuto di quanto è stato aggiunto viene trasferito effettivamente su file. Nel caso dell'esempio del programma di gestione del conto corrente, all'apertura, non era caricato alcun modulo o altro tipo di file; dopo avere creato il

modulo CC00 (che è il main), con il comando suddetto sono stati creati, nell'ordine, i moduli chiamati CC01, CC02, CC03, CC04, CC05, CC06, CCSU.

Tramite l'opzione **Unload File…** è invece possibile escludere dalla lista, e quindi liberare la memoria occupata, un file, sia esso un modulo, un file include o un file documento.

Se il file da aggiungere alla lista è già stato creato e lo vogliamo solo caricare in memoria, sia esso modulo, file include o documento, viene aperta una dialog box, del tipo di quella aperta dall'opzione Open Program…, che consente di scegliere tra i file contenuti su disco ed il tipo di file in questione.

2.2.5 Gli altri tipi di file

I file di tipo include contengono, come già detto, le dichiarazioni di SUB e FUNCTION, le frasi COMMON e tutto ciò che serve ad un modulo per funzionare correttamente. Le SUBs e le FUNCTIONs, dato che normalmente appaiono dopo il modulo a cui appartengono, devono essere dichiarate e, per evitare questo lavoro, che spesso è ripetitivo, viene dichiarato all'interno del modulo (generalmente all'inizio) il nome del file include tramite il metacomando $INCLUDE; in questo modo il file include è, a tutti gli effetti, facente parte del modulo ed è compilato in questa ottica.

Quindi, tutto ciò che è definito nel file include, vale anche per il resto del modulo ed è per questo che il metacomando $INCLUDE è sempre posto all'inizio del modulo stesso.

Nel caso dell'esempio dei conti correnti o in altri esempi che vedremo, sono stati creati due file include che contengono tutte le dichiarazioni delle SUBs e delle FUNCTIONs definite nella libreria BPLUS; sono inoltre contenuti dei buffers (aree di scambio dati con i file indicizzati) e dei dati generici che possono servire ad un qualsiasi programma (nomi dei giorni della settimana, nomi dei mesi) e delle costanti di uso generale.

Per quanto riguarda i file di tipo documento, non esiste molto da aggiungere a quanto già detto. Questi file vengono gestiti dal Quick Basic come farebbe un qualsiasi editor, senza alcun controllo o limitazione. Si possono caricare in memoria tramite il comando Load File… e si aggiungono alla lista dei moduli che costituiscono il progetto del programma, ma non ne fanno parte.

2.2.6 La stampa dei file

Vista l'organizzazione dell'ambiente del Quick Basic, i comandi LIST e LLIST, familiari al programmatore Basic che abbia programmato usando altre versioni di Basic, non sono necessari, l'uno perché il controllo della visualizzazione del testo è affidato ai tasti di editing (v. 2.3.1), l'altro perché è sostituito e migliorato dall'opzione **Print..** (**Stampa**) del menu File.
Questa operazione attiva una dialog box tramite la quale è possibile scegliere tra 4 modi differenti di stampare il testo con cui stiamo lavorando:

Selected Text Questo è il metodo per stampare solo le parti di testo che ci interessano dopo averle selezionate con gli appositi tasti (v. 2.3.2). È la via usata per stampare alcune linee di programma.

Active Window Dopo aver scelto questa opzione, viene stampato il contenuto della finestra attiva; questo contenuto può essere o il testo a livello modulo, o il testo di una SUB o quello di una FUNCTION.

Current Module È la modalità di default. Permette di stampare il contenuto di un intero modulo, quello attivo, comprese quindi le SUB e le FUNCTION eventualmente presenti. È il caso corrispondente al listato di tutto il programma se questo è fatto da un solo modulo (filosofia tradizionale di realizzazione dei programmi in Basic).

All Modules Con questa opzione vengono stampati tutti i moduli in memoria, i file di tipo include e di tipo documento eventualmente presenti in memoria. Viene stampato tutto ciò che viene evidenziato nella dialog box aperta dal tasto F2.

Se la stampante è pronta e collegata, la stampa avviene (la porta usata è la LPT1), altrimenti viene evidenziato un errore di tipo Device Fault o simili e l'operazione è abbandonata.

2.2.7 L'uscita da Quick Basic

Dal menu File è possibile scegliere due opzioni diverse che permettono di abbandonare l'ambiente di Quick Basic. La prima, **DOS Shell** (**DOS**), permette di ritornare temporaneamente al DOS per potere eseguire altri programmi o dei comandi dello stesso. Il Quick Basic ed i programmi che, eventualmente, erano già stati caricati, rimangono in memoria, ed al ritorno non è necessario ricaricarli. Per effettuare questa operazione, viene richiamato ed eseguito il COMMAND.COM che viene cercato, in un primo momento, nella directory indicata dalla variabile COMSPEC del DOS e, se non lo si è trovato, nella directory corrente.

Entrando in DOS, viene visualizzato da Quick Basic il messaggio 'Type EXIT to return to QuickBASIC', che indica appunto il comando (Exit) del DOS che è necessario immettere per tornare ad operare con il Quick Basic nel punto esatto dove lo stesso era stato abbandonato. Bisogna porre particolare attenzione nel compiere tale operazione alla quantità di memoria disponibile dato che ne viene occupata, solamente dal Quick Basic e dal Command, circa 230 K; a questa quantità va aggiunta quella del programma utente e della Quick Library che è possibile caricare.

Per abbandonare definitivamente Quick Basic e tornare in DOS, si usa l'opzione **Exit** (**Esci**). Se si fossero fatte delle modifiche al programma presente in memoria e queste non fossero ancora state registrate sul corrispondente file sul disco, il Quick Basic, al comando di uscita, aprirebbe una dialog box che permette di potere salvare la versione aggiornata del programma, di uscire senza aggiornare il file o di rinunciare all'uscita; nel caso in cui il programma non abbia ancora il nome, viene aperta la dialog box di cui al comando Save As…

2.3 IL MENU EDIT

2.3.1 Immissione e controllo del testo

Per immettere il testo di un programma, tramite l'editor che si ha a disposizione in ambiente Quick Basic, si deve semplicemente usare la sequenza di caratteri appropriata per comporre una linea, premere Return e continuare fino alla fine del testo stesso. L'uso del tasto Return non è necessario in quanto, dopo avere scritto una linea, è possibile spostare il cursore alla successiva dove si possono introdurre altre istruzioni. All'inizio, e ogni volta in cui il cursore ha la forma a cui siamo abituati, ci si trova in 'modo inserimento' (Inserting mode); i caratteri che digitiamo vengono inseriti all'interno della linea in cui ci si trova e, se dovesse essere presente del testo alla destra del cursore, questo sarebbe spostato a destra per fare posto ai nuovi caratteri. Per modificare questo stato di cose, il tasto Ins del tastierino numerico viene premuto una volta; il cursore cambia forma ed i caratteri digitati sono visualizzati al posto di quelli già presenti: si è in 'modo sovrascrittura' (Overtyping mode).

È abilitato, per default, il controllo delle istruzioni immesse nel testo (Syntax checking); alla digitazione del tasto Return o al momento del passaggio alla linea seguente con i tasti di controllo del cursore, viene effettuato dal Quick Basic un controllo di sintassi. Se quanto immesso non soddisfa le regole di sintassi generale del Quick Basic o è in contrasto con la sintassi particolare dell'istruzione indicata, viene emesso un messaggio di errore a video. Quando ciò avviene, con il tasto Return (o lo spazio, o Esc) è possibile cancellare questo messaggio e forzare l'errore nel testo, ma questo errore si ripresenterà al momento dell'esecuzione.

È possibile evitare che l'editor esegua il controllo della sintassi e ciò avviene tramite l'opzione **Syntax Checking** presente nel menu Edit (questo non è possibile con la versione 4.5). Dopo avere scelto tale opzione una volta, il controllo non viene più effettuato e gli errori saranno evidenziati soltanto al momento dell'esecuzione del programma; con un altro comando uguale, il controllo della sintassi viene riabilitato e tutto torna come prima.
Nel caso in cui si dovesse lavorare con dei file di tipo documento, il controllo della sintassi sarebbe automaticamente escluso dato che questi tipi di file sono trattati da Quick Basic come da un normale editor.

È da notare che, ad esempio, l'immissione della parola CMS (al posto della corretta istruzione CLS), non comporta un messaggio di errore da parte di Quick Basic; infatti, CMS può essere, e così viene interpretata, il nome di una SUB o di una FUNCTION a cui non devono essere passati dei parametri.

Oltre al controllo della sintassi, l'editor di Quick Basic esegue alcuni aggiustamenti automatici della linea immessa, e più precisamente:

* Tutte le parole chiave del linguaggio sono riscritte in lettere maiuscole

* Sono aggiunti, dove necessitano, degli spazi e tutti i segni di punteggiatura mancanti

* Il nome delle variabili e delle procedure è reso uguale in tutti i punti in cui esse sono usate (se in una parte del testo compare la variabile AliquotaIva, questa sarà nominata automaticamente in questa forma in ogni punto dove appare).

Per finire, all'atto dell'immissione della riga, se corretta, questa viene trasformata, in memoria, in codice oggetto direttamente eseguibile e questo è il punto di forza del compilatore Quick Basic.

2.3.2 Selezione del testo

Per permettere operazioni di cancellazione, spostamento o copia di porzioni di testo molto grandi, è possibile effettuare la selezione di una parte del testo che è interessata all'operazione in questione. Un testo selezionato viene evidenziato dalla luminosità dei caratteri inclusi in quest'ultimo, e ciò viene ottenuto, dalla posizione del cursore, con il tasto Shift e i tasti del controllo del cursore.
Ad esempio, per selezionare delle parole che stanno alla destra del cursore si preme lo Shift con il tasto di direzione destra, mentre, se si preme quello con la freccia verso il basso, si seleziona tutta la riga in basso, e così similmente per gli altri tasti.
Da notare, che il testo selezionato aumenta o diminuisce di ampiezza rispetto alle direzioni prese dal punto in cui si trovava il cursore all'inizio dell'operazione e che l'operazione stessa continua finché il tasto Shift viene tenuto premuto. Infatti, se si dovesse premere un tasto di controllo del cursore, senza tenere premuto lo Shift, il testo evidenziato dalla maggiore luminosità ritornerebbe normale e quindi non sarebbe più selezionato.
Altre combinazioni di tasti vengono usate per la selezione di altre parti di testo e queste sono elencate in seguito (v. 2.3.8).

2.3.3 La memoria di transito per l'edit (Clipboard)

Un'area di memoria particolare, detta Clipboard, viene utilizzata da Quick Basic per effettuare quelle operazioni di spostamento o copia del testo di cui si è già parlato (v. 2.3.2). In quest'area di memoria viene temporaneamente depositato il testo che deve essere spostato o copiato. Per permettere ciò, il testo in questione deve essere stato prima selezionato.

Tutte queste operazioni si effettuano attraverso delle opzioni del menu **Edit** (**Modifica**) o, in alternativa, con delle combinazioni di tasti che vengono indicati nel menu stesso a condizione che esista un testo selezionato (v. 2.3.8). Bisogna prestare attenzione al fatto che nella Clipboard il testo non si accumula; non è possibile cioè, selezionare due parti distinte di testo e copiarle dentro questa area perché il testo selezionato e trasferito per ultimo, prende il posto di quello eventualmente già presente nella Clipboard stessa.

2.3.4 Cancellazione, spostamento e copia del testo

Con il comando **Clear** (o con il tasto Del), il testo appena selezionato viene cancellato completamente dal file in memoria. Questo non può più essere recuperato se non rileggendo il file originale dal disco.

Il comando **Cut** (**Taglia**) (corrisponde alla combinazione di tasti Shift e Del), il testo selezionato viene prima trasportato nella Clipboard e poi cancellato dal testo. In seguito (con il comando Paste), è possibile recuperare tale testo in un'altra parte del programma e completare così lo spostamento del testo selezionato di partenza.

Tramite il comando **Copy** (**Copia**) (tasti Ctrl e Ins), il testo selezionato, come nel caso precedente, viene copiato all'interno della Clipboard, ma non viene cancellato dal testo. In seguito (con il comando Paste), il testo viene riscritto in un'altra zona del programma e viene completata l'operazione di copia del testo selezionato in precedenza.

Il comando **Paste** (**Incolla**) (tasti Shift ed Ins), serve a trasportare quanto memorizzato nella Clipboard, nel programma ad iniziare dalla posizione del cursore. È il comando finale delle operazioni di spostamento e copia.

Ad esempio, se il programma in memoria fosse fatto dalle seguenti linee:

```
DEFINT A-Z
CLS
FOR P=1 TO 100
  PRINT P
NEXT P
PRINT "IL VALORE CORRENTE È ";
```

e si volesse portare l'ultima all'interno del ciclo FOR..NEXT, si dovrebbe operare nel seguente modo:

a. Portare il cursore all'inizio dell'ultima linea e premere i tasti Shift ed End per selezionare tutta la riga (il testo selezionato diventa luminoso).

b. Premere il tasti Alt ed E per aprire il menu Edit; premere poi il tasto T (comando Cut) per eliminare dal testo la linea in questione e trasferirla nella Clipboard (si poteva effettuare la stessa operazione dopo avere selezionato il testo, premendo i tasti Shift e Del che effettuano il comando Cut).

c. Spostare il cursore sulla linea vuota all'interno del ciclo FOR…NEXT sopra la lettera iniziale della linea seguente e premere i tasti Alt ed E per aprire il menu Edit; premere a questo punto il tasto P per eseguire il comando Paste; la linea selezionata in precedenza viene trasferita nella posizione voluta. Il testo assume la seguente forma:

```
DEFINT A-Z
CLS
FOR P=1 TO 100
  PRINT "IL VALORE CORRENTE È ";
  PRINT P
NEXT P
```

2.3.5 Il comando Undo

Dopo aver effettuato delle modifiche in una linea di testo, con il comando **Undo** (**Annulla**) del menu Edit (**Modifica**) (o con i tasti Alt e Backspace), si può fare tornare una linea come era in origine, come se le modifiche non fossero mai state fatte. Questo comando è eseguibile solo se

il cursore non è stato mosso dalla linea interessata, in quanto, nel caso in cui lo si fosse spostato in altre linee, le modifiche fatte sarebbero diventate permanenti.

Non è possibile con questo comando recuperare le linee se queste sono state cancellate con i tasti Ctrl e Y.

2.3.6 Immissione di simboli speciali

Per immettere tutti i caratteri corrispondenti ai codici ASCII che vanno dal 128 al 255 è sufficiente immettere il valore corrispondente sul tastierino numerico tenendo premuto il tasto Alt.

Per quanto riguarda i caratteri corrispondenti ai codici che vanno dal numero 1 al 27, essendo questi interpretati come comandi (ad esempio il 25, Ctrl Y, cancellazione di una riga), saranno immessi con un'altra procedura. Si deve premere la combinazione di tasti Ctrl e P e solo a questo punto si può premere il tasto Ctrl con il simbolo associato al valore del carattere ASCII da visualizzare. Ad esempio, per visualizzare i simboli dei semi delle carte francesi, si farà:

Ctrl P C	appare il simbolo del seme di cuori
Ctrl P D	appare il simbolo del seme di quadri
Ctrl P E	appare il simbolo del seme di fiori
Ctrl P F	appare il simbolo del seme di picche

Non è possibile agire in questa maniera solo per i simboli associati ai tasti J, L, M ed U; per questi (come per gli altri sarebbe più elegante), si può usare la funzione CHR$.

2.3.7 Le opzioni New SUB… e New FUNCTION…

Quando si deve creare una nuova SUB all'interno di un modulo, si sfrutta l'opzione **New SUB…** (**Nuova SUBroutine**) del menu Edit (**Modifica**). Viene aperta una dialog box apposita con cui viene richiesto il nome della SUB e, dopo la digitazione del tasto Return, viene creata l'apposita finestra che dovrà contenere la sottoroutine con le frasi SUB <nome sub> ed END SUB introdotte automaticamente. Lo stesso avviene per le FUNCTION quando si usa l'opzione **New FUNCTION…** (**Nuova Function**), con la differenza che Quick Basic introduce le frasi FUNCTION <nome function> ed END FUNCTION.

Il <nome sub> e il <nome function> può avere una lunghezza massima di 40 caratteri.

Un modo più veloce per creare nuove SUB o FUNCTION è quello di scrivere, in qualsiasi parte del modulo, la frase di apertura delle strutture (SUB <nome sub> o FUNCTION <nome function>); dopo l'uso del tasto Return, automaticamente Quick Basic porta il cursore e la frase scritta all'interno della finestra creata appositamente a contenere il testo.

Le SUB o le FUNCTION fanno parte sempre di un modulo; vengono salvate sul disco insieme ad esso e vengono rilette quando questo viene ricaricato in memoria. Ciò non vuol dire che altri moduli, caricati insieme (programma multi-modulo), non possa usare le stesse SUB o FUNCTION definite in un altro; anzi, è questa la caratteristica fondamentale di queste strutture e per fare

ciò è solo necessaria la frase di dichiarazione delle SUB o delle FUNCTION nei moduli che le usano (DECLARE v. 5.1.2).

2.3.8 Tabella di riepilogo dei tasti di edit

In questa tabella sono riassunti tutti i tasti e le combinazioni di tasti, con le relative funzioni, che l'editor di Quick Basic mette a disposizione dell'utente. Il simbolo ^ indica che deve essere premuto il tasto Ctrl e tra parentesi viene indicato un tasto o la combinazione di tasti alternativa alla principale.

POSIZIONAMENTO DEL CURSORE

- Carattere a destra	**Tasto curs. dx (^D)**
- Carattere a sinistra	**Tasto curs. sx (^S)**
- Linea in alto	**Tasto curs. alto (^E)**
- Linea in basso	**Tasto curs. basso (^X)**
- Parola a destra	**^Tasto curs. dx (^F)**
- Parola a sinistra	**^Tasto curs. dx (^A)**
- 1^ livello di indent.	**Home**
- Col. 1 della linea corrente	**^QS**
- Inizio della prossima linea	**^Return (^J)**
- Fine linea	**End (^QD)**
- Inizio della finestra	**^QE**
- Fine della finestra	**^QX**
- Inizio del modulo/procedura	**^Home (^QR)**
- Fine del modulo/procedura	**^End (^QC)**

INSERIMENTO

- Attiva/Disattiva modo inserimento	**Ins (^V)**
- Una linea prima	**End e poi Return**
- Una linea dopo	**Home e poi ^N**
- Il contenuto della Clipboard	**Shift e Ins**
- Gli spazi di una tabulazione	**Tab (^I)**

CANCELLAZIONE

- Il carattere sotto il cursore	**Del (^G)**
- Il carattere a sinistra	**Backspace (^H)**
- La linea corrente dopo averla salvata nella Clipboard	**^Y**
- Fino alla fine della linea dopo avere salvato questa parte nella Clipboard	**^QY**
- Una parola	**^T**
- Il testo selezionato senza averlo salvato nella Clipboard	**Del (^G)**
- Il testo selezionato dopo averlo	

salvato nella Clipboard	**Shift e Del**
- Gli spazi a sinistra fino al precedente livello di indentazione	**Shift e Tab**

SELEZIONE DEL TESTO

- Un carattere a sinistra	**Shift e Tasto c. sx**
- Un carattere a destra	**Shift e Tasto c. dx**
- La linea corrente	**Shift e Tasto c. basso**
- La linea precedente	**Shift e Tasto c. alto**
- Una parola a sinistra	**Shift e ^Tasto c. sx**
- Una parola a destra	**Shift e ^Tasto c. dx**
- Una schermata in alto	**Shift e Pgup**
- Una schermata in basso	**Shift e Pgdn**
- Una schermata a sinistra	**Shift e ^Pgup**
- Una schermata a destra	**Shift e ^Pgdn**
- Fino all'inizio del mod./proc.	**Shift e ^Home**
- Fino alla fine del mod./proc.	**Shift e ^End**

SCORRIMENTO

- Una linea in alto	**Tasto c. alto (^W)**
- Una linea in basso	**Tasto c. basso (^Z)**
- Una pagina in alto	**Pgup (^R)**
- Una pagina in basso	**Pgdn (^C)**
- Una schermata a sinistra	**^Pgup**
- Una schermata a destra	**^Pgdn**

COPIA DEL TESTO

- Il testo selezionato nella Clipboard	**^Ins**

2.4 IL MENU SEARCH

2.4.1 La ricerca del testo

Quick Basic mette a disposizione dell'utente dei potenti comandi per la ricerca di parti di testo all'interno del programma. È normale infatti che, specialmente per programmi multi-modulo molto grandi, si ci sposti frequentemente all'interno del testo per effettuare modifiche alle linee, e questo comporta parecchio spreco di tempo se si deve cercarle senza alcun automatismo. Con l'opzione **Find…** (**Trova…**) del menu Search (**Ricerca**), è invece possibile rintracciare qualsiasi parte di testo all'interno di un programma e ciò concorre ad abbreviare drasticamente i tempi di sviluppo del software.

Nella dialog box che viene aperta deve essere indicato il testo cercato ed alcune opzioni di ricerca:

Active Window Ricerca soltanto nella finestra attuale, sia essa quella di una SUB, di una FUNCTION o del testo a livello del modulo.

Current Module Ricerca in tutto il modulo attivo sia nel testo del livello modulo sia nel testo di tutte le SUB e le FUNCTION che ne fanno parte. È il modo di default.

All Modules È il modo per ricercare in tutti i moduli presenti in memoria. È usato quando, all'interno di un programma multi-modulo, si debba cercare un testo che può non essere nel modulo corrente.

Match Upper/Lowercase Se viene attivato questo parametro, la ricerca tiene conto delle lettere maiuscole e minuscole. Normalmente il testo cercato viene evidenziato anche se viene trovato con il set di caratteri diversi (ad esempio, normalmente, se si cerca IVA = 19 si può trovare anche Iva = 19).

Whole Word Se questo parametro è attivo, la ricerca avviene sottointendendo che la parola cercata deve essere trovata isolata nel testo e cioè preceduta e seguita da quei caratteri, come lo spazio o il punto, che la fanno identificare appunto come parola (ad esempio, se si cerca, con questa opzione attivata, la parola IVA, non può essere trovata all'interno della parola ALIQUOTAIVA).

Se il testo cercato non è stato trovato, un messaggio adeguato (Match not found) viene evidenziato a video.

Per ripetere la ricerca dell'ultimo testo cercato, esiste l'opzione **Repeat Last Find** (**Ripeti Trova**); è più comodo usare il tasto F3 che è appositamente usato per questa funzione.

L'opzione **Selected Text** (**Testo selez.**) è del tutto simile alla Find…, ma è utile quando si debba cercare il testo precedentemente selezionato tramite i tasti di selezione; questo deve essere non più grande di una linea. Se il testo non è stato selezionato, viene aperta la stessa dialog box del comando Find… e viene proposta la parola su cui è posizionato il cursore. Il comando può essere anche eseguito tramite i tasti Ctrl e \.

L'opzione **Label** (**Etichetta**) serve a cercare nel testo le etichette a cui possono fare riferimento un gruppo di istruzioni (ad esempio, GOTO, GOSUB, RETURN e RESTORE) e che devono terminare sempre con il simbolo dei due punti (:). Può essere cercata la parola intera (default) o una sua parte, ma deve essere sempre una parte di etichetta.

2.4.2 Ricerca e modifica del testo

È presente anche una opzione, la **Change…** (**Sostituisci**), che permette di cercare una parte di testo nel programma e di sostituirlo con un altro automaticamente. La dialog box che viene aperta in questo caso comprende anche la richiesta del testo da sostituire oltre che tutti i parametri già visti nel comando Find…

La ricerca e la sostituzione può essere automatica o, come è per default, ogni volta che il testo viene trovato, si può decidere se confermare la sostituzione e continuare la ricerca, se continuare la ricerca senza sostituire il testo trovato o se abbandonare l'operazione. Questa

opzione rende molto potente l'editor di Quick Basic dato che è possibile agire, come per il comando Find…, anche su programmi multi-modulo.

Prestare particolare attenzione all'opzione che cambia il testo, specialmente se in modo automatico (Change All), perché l'operazione di cambio del testo non è reversibile con il comando Undo.

2.4.3 Uso dei marcatori di testo

Se si sta lavorando in una parte del programma e ci si deve spostare in un'altra temporaneamente per poi tornare alla precedente, è il caso di usare i **marcatori di testo**. Questi sono dei 'segni' che Quick Basic lascia in una particolare posizione del testo in modo da poi potere ritornare velocemente in quella posizione, da qualsiasi altro punto del testo stesso.

Si possono marcare fino a 4 posizioni distinte per volta, numerate da 0 a 3, e per fare ciò viene adoperata la combinazione dei tasti Ctrl K n, in cui n è il numero (da 0 a 3) del segno da piazzare nella posizione del cursore.

Per ritrovare i vari segni lasciati, si adopera la combinazione dei tasti Ctrl Q n, dove n è sempre il numero del segno a cui si vuole tornare.

Questo modo di operare è sicuramente familiare a chi usa correntemente il Wordstar.

2.4.4 Tabella di riepilogo dei tasti di ricerca

Questa tabella riassume i tasti e le combinazioni di tasti, con le funzioni associate, che servono per la ricerca dei testi in Quick Basic. Il simbolo ^ indica che deve essere usato il tasto Ctrl.

- Ricerca del testo selezionato	**^**
- Ripete l'ultima ricerca	**F3**
- Segna il punto n(*) nel testo	**^Kn**
- Cerca il punto n(*) nel testo	**^Qn**

(*) il numero n può andare da **0** a **3**

2.5 IL MENU VIEW

2.5.1 Lo schermo di output

Come già accennato, Quick Basic usa due pagine video diverse, una per visualizzare il programma e i suoi menu e l'altra per visualizzare i dati dell'utente. Quest'ultima pagina video viene chiamata Output Screen ed è quella a cui tutte le istruzioni di visualizzazione del Quick Basic si riferiscono. Per cambiare temporaneamente la pagina video visualizzata, si può scegliere l'opzione **Output Screen** (**Schermo output** nel menu **Visualizza**) o, in alternativa, e molto più semplicemente, agire sul tasto F4. Questo tasto si comporta come un commutatore; premuto più volte, permette di visualizzare alternativamente, la pagina video utente e quella riservata all'ambiente di Quick Basic.

2.5.2 La gestione delle SUB e delle FUNCTION

Tramite l'opzione **SUBs…** (**SUBroutine**) del menu View (**Visualizza**) (o con il tasto F2), si apre una dialog box con cui potere gestire i moduli e le procedure che compongono un programma. Evidenziate, dalla luminosità e dalla posizione all'interno della dialog box, le procedure a livello modulo (v. 2.2.4) sono seguite dalla SUB e dalle FUNCTION che le completano. Per visualizzare ed editare una SUB o una FUNCTION all'interno di un modulo, o per modificare uno dei testi a livello modulo, basta portare il cursore in basso, con il tasto di spostamento adatto, e premere Return (comando di Edit nella Active Window).

Per visualizzare sullo schermo, contemporaneamente, i testi di due SUB diverse, dopo avere selezionato la prima nel modo già indicato, bisogna tornare a premere F2, scegliere la seconda SUB da visualizzare e selezionare, all'interno della dialog box corrente, l'opzione Split. Con l'uso ripetuto del tasto F6, potremo abilitare a turno una delle finestre e quindi si potranno editare le due SUB insieme. È possibile che parti di testo di una vengano spostate nell'altra, con i comandi del menu Edit (v. 2.3.4) in maniera rapida e semplice.

Per far tornare lo schermo nella modalità normale dopo avere lavorato con due testi diversi, o, in alternativa al primo metodo, per dividere lo schermo in due parti, è sempre disponibile il comando **Split** (**Divisione finestra**) del menu **View** (**Visualizza**). Esso agisce come un interruttore, permettendo il cambio della modalità con cui Quick Basic gestisce il video, in maniera molto semplice.

Le stesse operazioni sono valide nel caso in cui si tratti di **FUNCTION**, o di qualsiasi altro tipo di testo che è possibile caricare dal disco con il comando Load del menu File (v. 2.2.4).

Per eliminare dalla memoria una sola SUB o una FUNCTION, si deve operare il comando Delete trovandosi all'interno della dialog box aperta con il tasto F2. La SUB o la FUNCTION cancellata sarà quella su cui avremo portato il cursore. Viene richiesta una conferma prima di eliminare la parte di testo prescelto dal modulo.

Per ultimo è possibile spostare automaticamente una SUB o una FUNCTION, da un modulo ad un altro, in modo che la gestione dei programmi multi-modulo, sia più semplice e razionale. Per fare ciò basta portare il cursore sull'elemento che si vuole spostare e, dopo avere dato il comando Move, si sceglie il modulo a cui si desidera che la SUB o la FUNCTION venga aggiunta.

Per un più veloce spostamento all'interno delle procedure di un modulo, si può usare l'opzione **Next SUB** (**SUB succ.**) (o la combinazione di tasti Shift e F2), che permette di editare la prossima SUB del modulo corrente, rispetto a quella già visualizzata.

2.5.3 Gestione dei file include

Se si volesse visualizzare il contenuto di un file include di cui esiste il metacomando relativo all'interno di un programma, il menu View mette a disposizione l'opzione **Included Lines** (**Righe incluse**) che permette di evidenziare, a partire dalla posizione del metacomando nel testo principale, le linee del file include stesso; questa caratteristica è valida per tutti i file include di cui esiste il metacomando nel testo principale.

Le linee del file include vengono evidenziate in alta luminosità, ed una volta attivato il comando, è solo possibile scorrere il testo per controllarlo, ma nessuna correzione è consentita. Se si tentasse di apportare delle correzioni al testo del programma mentre le linee del file include sono visualizzate, si otterrebbe un messaggio di avvertimento che vieta l'operazione descritta; ma, attraverso una dialog box, verrebbe data la possibilità di eliminare dal testo le linee del file include per potere effettuare l'editing del programma.
Se, invece, si tentasse di editare le linee stesse del file include, allora verrebbe aperta un'alta dialog box che, con dei messaggi appropriati, permetterebbe di caricare in memoria il file e modificare le sue linee.

Per fare ciò esiste anche un comando apposito, sempre nel menu View: **Include File** (**File incluso**). Questa opzione è usata per caricare ed editare direttamente, senza passare dalla fase di visualizzazione precedentemente descritta, di un file include compreso nel testo principale. L'operazione si effettua solamente se il cursore è posto sulla linea dove compare il metacomando ($INCLUDE) relativo al file include che si vuole editare; in caso contrario, infatti, l'opzione appare nel menu View, in un altro colore che sta appunto ad evidenziare il fatto che essa non è operante.

I file include che si fossero caricati e modificati con le operazioni precedenti, devono essere salvati sul disco, e per questo, è comodo usare l'opzione Save All del menu File (v. 2.2.2).

2.5.4 Altre opzioni del menu View

Dato che il comando **Options…** è stato già esaminato nel precedente capitolo (v. 1.2.2), rimane da descrivere il funzionamento dell'ultima opzione rimanente: **Next Statement** (**Istruzione succ.**). Tramite questo comando è sempre possibile tornare a visualizzare nella finestra attiva, ovunque ci troviamo, la linea che contiene le prossime istruzioni da eseguire che, se il programma non è stato mai eseguito o dopo un Restart, corrisponde alla prima linea del Main Module. È importante avere a disposizione questa possibilità in quanto è normale che, specialmente durante una sessione di lavoro con un programma multi-modulo, ci si sposti continuamente attraverso il testo del programma, perdendo così di vista il punto in cui si è interrotta, per diversi motivi, l'esecuzione del programma stesso.

CAPITOLO 3

Creazione dei programmi

CAPITOLO 3

3.1 L'AMBIENTE DI SVILUPPO DEI PROGRAMMI

3.1.1 Uso della Immediate Window

La Immediate Window è una parte della schermata di Quick Basic in cui è possibile eseguire direttamente delle istruzioni del linguaggio, al di fuori del contesto di un qualsiasi programma caricato in memoria. Nel BASICA e nel GWBASIC è paragonabile al modo diretto anche se, come si vedrà in seguito, l'uso della Immediate Window comporta importanti vantaggi.

Si possono eseguire direttamente delle linee lunghe fino a 256 caratteri; le istruzioni così raggruppate devono essere separate dall'adeguato segno (:). Ogni linea è eseguita indipendentemente dall'altra anche se non è stata scritta in ordine e per fare ciò basta premere il Return dopo aver posto il cursore sulla linea prescelta. Le dimensioni della finestra possono essere variate, dopo averla selezionata con il tasto F6, usando i tasti Alt + ed Alt -; di contro i tasti Ctrl e F10 ampliano la Immediate Window fino alla dimensione totale dello schermo. Le istruzioni scritte in questa finestra, eseguite e non, non sono registrate su disco, ma possono essere trasferite usando i comandi di edit adeguati, nella finestra in cui è scritto il testo del programma e salvate con esso.

Non tutte le istruzioni di Quick Basic possono essere eseguite nella Immediate Window e quella che segue è appunto la lista di quelle vietate:

COMMON, CONST, DATA, DECLARE, DEF FN, DEFtype, DIM, $DYNAMIC, ELSEIF, END DEF, END FUNCTION, END IF, END SUB, END TYPE, FUNCTION, $INCLUDE, OPTION, REDIM, SHARED, $STATIC, SUB, TYPE

È quindi possibile, per esempio, testare il contenuto di una variabile in un determinato momento dell'esecuzione di un programma usando l'istruzione PRINT apposita e controllando il risultato nell'Output Screen attraverso il tasto F4.

Le SUB e le FUNCTION definite all'interno di un programma possono essere altresì eseguite nell'Immediate Window per controllarne gli effetti isolate dal programma stesso.

Il contenuto delle variabili infine, può essere cambiato durante l'esecuzione di un programma, se ciò può essere utilizzato per testare parti dello stesso.

3.1.2 Immissione delle istruzioni di un programma

Per immettere le istruzioni di un programma in Quick Basic è utilizzato l'editor di cui esso dispone, le cui potenzialità sono già state trattate in precedenza.

Le linee di istruzioni sono immesse semplicemente ed analizzate dopo l'uso del tasto Return; se la loro sintassi è corretta esse vengono tradotte in codice oggetto e conservate in memoria, mentre se l'uso che se ne fa non rispetta la sintassi dettata dal linguaggio, viene aperta una dialog box che informa il programmatore, in maniera dettagliata, del tipo di errore. Ad esempio, se la linea

IF K>45

fosse stata confermata con il tasto Return, Quick Basic avrebbe evidenziato in una dialog box la mancanza della parola chiave THEN che non è stata aggiunta ma che lui si aspettava. Ecco quindi che la segnalazione dell'errore di sintassi diventa più evoluta e facilita il lavoro del programmatore.

Nel caso in cui la sintassi di un particolare comando non fosse chiara o sfuggisse al momento al programmatore, la possibilità di usate un vero e proprio manuale in linea attraverso il menu di aiuto, rende ulteriormente agevole l'uso del linguaggio, sia nella versione 4.0 che, ancora maggiormente nella 4.5 in cui il manuale è veramente disponibile per intero da programma.

Nel caso in cui si confermasse una istruzione errata ignorando il messaggio di errore di Quick Basic, la stessa verrebbe conservata nel testo ma l'errore si ripresenterebbe al momento dell'esecuzione dello stesso. Il controllo della sintassi di ciò che è contenuto in una linea viene effettuato ogni volta che si abbandona una linea con il cursore dopo averne modificato almeno un carattere.

3.1.3 Creazione di un Main Module

In un programma di tipo singolo-modulo il Main Module è il programma stesso. Quando si immettono le prime istruzioni di un programma queste costituiscono il Main Module; esse sono le prime che vengono eseguite al momento della partenza. Insieme alle istruzioni facenti parte degli altri moduli esse costituiscono un programma di tipo multi-modulo. Per chiarire questo concetto, ecco rappresentati i tipi di programmi realizzati in Quick Basic:

*** Programma singolo-modulo**

```
Main module
      [ Subs ]
      [ Functions ]
```

*** Programma multi-modulo**

```
Main module
      [ Subs ]
      [ Functions ]
   Altri moduli
        [ Subs ]
        [ Functions ]
```

Il nome del Main Module corrisponde a quello che appare al centro della schermata nella barra superiore accanto alla scritta Main. È il nome con cui viene salvato il testo sul disco e all'inizio della sessione di lavoro è Untitled.

3.1.4 Creazione di un programma multi-modulo

Quando viene caricato in memoria da Quick Basic un file di programma questo è inteso come un **modulo**. Un programma può essere fatto da più moduli oppure un singolo modulo può costituire un programma: nel primo caso si parlerà di programmi multi-modulo, nel secondo di programmi singolo-modulo (e cioè quelli a cui la maggior parte dei programmatori Basic sono abituati). È comunque il Main Module quello che contiene le prime istruzioni da eseguire mentre gli altri moduli, se esistono, sono richiamati in una sequenza determinata dal programmatore.

Per creare un modulo all'interno di un programma multi-modulo basta servirsi dell'opzione Create del menu File con cui viene aperta una dialog box che permette di immettere il nome del nuovo modulo da creare. Con la stessa opzione si possono creare file documento o file include ma per default, l'indicatore è posizionato in maniera da creare nuovi moduli.

Una volta dato il nome al modulo, questo viene selezionato ed è possibile scrivere le istruzioni in esso contenute. Per avere la situazione sotto controllo è sufficiente premere il tasto F2 tramite il quale vengono evidenziati in una dialog box tutti i nomi dei moduli presenti, le loro eventuali Subs e Functions ed il nome del Main Module; tramite la stessa dialog box è possibile spostarsi all'interno dei vari moduli, Subs e Functions.

Per esempio, per creare un programma di prova formato da due moduli ecco i passi da seguire:

* all'apertura di Quick Basic immettere le seguenti linee di programma che costituiranno il Main Module

```
DECLARE FUNCTION PERIMETRO (L)
DECLARE FUNCTION AREA (L)
CLS
INPUT "Lato del quadrato : ", LATO
PRINT "Perimetro : "; PERIMETRO (LATO)
PRINT "Area : "; AREA (LATO)
```

* Dopo avere selezionato l'opzione Create del menu File tramite i tasti Alt F C, immettere il nome del secondo modulo costituente il programma, modulo che conterrà le functions definite nel Main Module e che si chiamerà QUADRATO.

Al tasto Return il video si pulirà e sarà possibile immettere il contenuto delle due functions nel modulo appena creato.
Premendo F2 si potrà vedere il tipo di lavoro effettuato; saranno infatti presenti il Main Module ancora Untitled (dato che non gli abbiamo dato ancora alcun nome) ed il modulo QUADRATO. Dopo aver premuto il tasto Esc per tornare all'editing del modulo QUADRATO, per inserire la function PERIMETRO basta scegliere l'opzione New Function… del menu Edit con i tasti Alt E F ed immettere il nome della function (PERIMETRO) alla richiesta della relativa dialog box. Dato il tasto Return, si immetterà l'unico argomento della funzione nella testata della stessa così da avere il seguente testo:

```
FUNCTION PERIMETRO (LATO)
END FUNCTION
```

Lo stesso verrà modificato nel seguente modo:

```
FUNCTION PERIMETRO (LATO)
        PERIMETRO=LATO*4
END FUNCTION
```

ed alla stessa maniera verrà introdotta la function relativa all'area:

```
FUNCTION AREA (LATO)
        AREA=LATO*LATO
END FUNCTION
```

A questo punto, premendo il tasto F2, ci verrà evidenziata la situazione che vede la presenza di due moduli (Untitled che è il Main e QUADRATO) e di due Functions (PERIMETRO ed AREA) contenute nel modulo QUADRATO. Si potrà salvare il tutto con l'opzione Save All del menu File che, appunto, serve a registrare su disco i programmi multi-modulo; a questo punto verrà richiesto il nome da dare al Main Module tramite la relativa dialog box, e noi immetteremo ESEMPIO.

A comando eseguito, sul disco saranno presenti i seguenti file:

```
ESEMPIO.BAS
QUADRATO.BAS
ESEMPIO.MAK
```

I primi due file conterranno i moduli corrispondenti nel programma (le functions saranno in QUADRATO.BAS perché appartenenti a quel modulo) mentre il file ESEMPIO.MAK conterrà delle informazioni riguardanti la composizione del programma multi-modulo.

3.1.5 Il file .MAK

I file con estensione .MAK non sono altro che dei semplici file testo in cui Quick Basic scrive i nomi dei moduli che compongono un programma multi-modulo. Nel caso precedente, ad esempio, il file ESEMPIO.MAK contiene le due linee seguenti:

```
ESEMPIO.BAS
QUADRATO.BAS
```

che non sono altro che i nomi dei due file nei quali sono stati registrati i due moduli costituenti il programma. Il primo dei due è il Main Module.

Questo tipo di file costituisce il **progetto** di un programma multi-modulo e Quick Basic ne controlla la presenza per permettere di ricaricare tutti i moduli esattamente come sono stati salvati per editare il programma multi-modulo. Se si volesse ricaricare il programma ESEMPIO si potrebbe operare nei seguenti due modi:

```
QB ESEMPIO
```

oppure

QB ESEMPIO.MAK

In ambedue i casi, tramite il file ESEMPIO.MAK, viene ricaricato tutto il programma multi-modulo; ciò non sarebbe stato possibile dando il comando QB QUADRATO che avrebbe caricato in memoria il file QUADRATO.BAS intendendo quest'ultimo come Main Module di un programma singolo-modulo.

3.1.6 Cambio del Main Module

Per potere selezionare un Main Module diverso in programmi multi-modulo, quando se ne presenti la necessità, esiste l'opzione **Set Main Module… (Imposta modulo princ.**) del menu **Run** (**Esegui**). Dopo che viene aperta una dialog box che contiene tutti i nomi dei moduli presenti nel programma, tramite i tasti di controllo del cursore ed il tasto Return è possibile scegliere il nuovo Main Module.

3.1.7 Gestione dei moduli in un programma multi-modulo

All'interno di un programma multi-modulo è possibile gestire i moduli in maniera da visualizzarne il contenuto su una parte di schermo, di spostare le Subs e le Functions tra moduli diversi e di eliminarne interi in un solo colpo.

Tramite il tasto F2 si apre una dialog box che permette di scegliere quale modulo deve essere visualizzato nella finestra attiva (**Active Window**) e quale deve essere invece inviato alla finestra alternativa (**Split Window**) e ciò è possibile usando i tasti di posizionamento del cursore per scegliere un modulo, del tasto di tabulazione per scegliere tra le due finestre e del tasto Return per confermare l'operazione.

Per esempio, dopo avere caricato il programma multi-modulo ESEMPIO, si può scegliere di visualizzare il Main Module nella finestra attiva e la function AREA nella finestra alternativa dopo avere selezionato tale function ed avere scelto la Split Window. Con il tasto F6 ci si può muovere tra le finestre suddette in maniera da tenere presenti diverse parti di codice per una più efficiente realizzazione dei programmi.

Per potere spostare delle Subs o delle Functions da un modulo ad un altro, è utilizzabile l'opzione Move presente all'interno della dialog box aperta dal tasto F2; per fare ciò si seleziona con i tasti cursore la Sub o la Function da spostare, ci si porta con il tasto Tab sull'opzione Move e si preme Return; viene aperta un'altra dialog box tramite la quale si può scegliere, usando i tasti cursori, il modulo in cui le Subs o le Functions selezionate devono essere spostate.

Infine, per eliminare un modulo, come per una Sub o una Function, basta selezionare l'oggetto desiderato, portarsi con il tasto Tab sull'opzione Delete della dialog box aperta dal tasto F2 e questo viene eliminato dalla lista.

È naturalmente possibile effettuare tutte queste operazioni servendosi del mouse al posto della tastiera.

3.2 IL MENU RUN

3.2.1 Esecuzione dei programmi

Dopo avere scritto un programma, sia singolo-modulo che multi-modulo, o dopo averne caricato uno da disco, tramite l'opzione **Start** (**Avvia**) del menu Run è possibile farlo eseguire da Quick Basic. Lo stesso effetto, anche se in maniera più pratica e veloce, si ottiene premendo i tasti Shift e F6; l'esecuzione è quasi istantanea dato che il processo di compilazione è già stato eseguito nella sua massima parte. È a questo punto che Quick Basic controlla che tutte le chiamate a delle Subs o a delle Functions siano valide e che non ci siano errori con tutte quelle strutture di controllo, come IF…END IF, DO…LOOP, SELECT CASE…END SELECT, FOR…NEXT, che hanno un inizio ed una fine.

Solo alla fine del programma o se questo viene interrotto, con una istruzione di STOP o con i tasti Ctrl Break, Quick Basic visualizza nuovamente lo schermo contenente il programma, mentre normalmente durante l'esecuzione dello stesso viene visualizzato lo schermo di Output (quello dell'utente ottenibile con il tasto F4). Ad una interruzione del programma, viene evidenziata l'istruzione alla quale lo stesso è stato bloccato; è possibile a questo punto, controllare il contenuto di ogni variabile utilizzata, servendosi della Immediate Window, al fine di controllare eventuali errori logici. Per riprendere l'esecuzione dal punto in cui era avvenuta l'interruzione basta usare l'opzione **Continue** (**Continua**) del menu Run (Esegui) o basta premere il tasto F5.

Alcune volte, specialmente durante il debugging (v. 3.3.1) è necessario azzerare tutte le variabili e riportare l'esecuzione del programma all'inizio della prima istruzione per poi testarlo riga per riga e ciò è possibile con l'opzione **Restart** (**Riavvia**) del menu Run.

3.2.2 La variabile di sistema COMMAND$

Quando un programma, scritto in Quick Basic, viene compilato e trasformato in un file EXE eseguibile da MS-DOS, può essere fatto partire semplicemente invocandolo da DOS; è sempre possibile passare allo stesso dei parametri sotto forma di parole e simboli e ciò si ottiene facendo seguire gli stessi al nome del programma. Ad esempio, per fare in modo che i parametri /Q e /K LIBA possano passare al programma PROVA.BAS dopo che lo stesso è stato compilato, basta chiamare da DOS il programma con la seguente linea:

PROVA /Q /K LIBA

Il Quick Basic passa tutti i dati presenti dopo il nome del programma ad una variabile interna riservata chiamata **COMMAND$**. Esaminando il contenuto di tale variabile è possibile elaborare i parametri passati dall'esterno. La stessa cosa è possibile se il programma PROVA è eseguito nell'ambiente QB, scrivendo:

(1) QB /RUNPROVA /CMD / Q /K LIBA

e tenendo presente che l'opzione /CMD del comando QB deve essere l'ultima nella riga di chiamata.
Ad esempio, il seguente programma,

```
CLS
PRINT COMMAND$
END
```

fatto eseguire come nella riga **(1)**, scrive sul video la seguente frase:

```
/Q /K LIBA
```

che non è altro che il contenuto della variabile COMMAND$. Se per provare il programma, adesso si volesse modificare il contenuto di tale variabile, come se si fosse data una diversa linea richiamando il programma, si usa l'opzione **Modify COMMAND$...** (**Modifica COMMAND$...**), del menu Run; viene aperta una dialog box in cui è possibile immettere ciò che si vuole caricare nella variabile COMMAND$ e dare Return; a questo punto si può rieseguire, con i tasti Shift F5, il programma per ottenere l'effetto voluto.
Tale opzione è utilissima nel realizzare programmi di utilità che, una volta compilati, accettino dei parametri dalla linea di comando di MS-DOS.

3.2.3 Creazione di file .EXE

Una volta che il programma è stato rivisto e corretto e si voglia creare un file .EXE frutto della compilazione dello stesso, in modo da poterlo eseguire da MS-DOS, si può usare l'opzione **Make EXE File…** (**Crea un file EXE**) del menu Run che fa apparire una dialog box con cui potere scegliere alcuni parametri; viene indicato il nome che avrà il file .EXE, che normalmente è quello del file .BAS, ed un volta accettatolo con Return, si possono scegliere altre opzioni:

* codice per il debug

viene richiesto se il file .EXE deve contenere dei dati utili in seguito per la rilevazione degli errori run-time;

* file .EXE che richiede il BRUN40.EXE o di tipo stand-alone

si deve scegliere se creare un file .EXE che non ha bisogno del BRUN40.EXE per la sua esecuzione (tipo stand-alone) o che ne ha bisogno (tipo normale); il tipo stand-alone è sempre di dimensioni maggiori perché contiene delle informazioni che normalmente vengono fornite dal file BRUN40.EXE;

* continuazione dell'editing

si può scegliere che dopo la compilazione e la realizzazione del file .EXE, si continui nell'editing del programma o che si esca da Quick Basic una volta che la compilazione è terminata.

Una volta date tutte queste informazioni, Quick Basic lancia automaticamente il programma compilatore di linea BC che crea il file oggetto (.OBJ) ed il programma LINK che, dal file oggetto, crea il file eseguibile (.EXE); informazioni più dettagliate circa il funzionamento di questi due programmi e sulla realizzazione dei file .EXE sono presenti nel Capitolo 4.

3.2.4 Creazione delle Quick Libraries

Anche per la creazione delle Quick Libraries è presente una opzione nel menu Run (**Make Library…** (**Crea libreria**)) tramite la quale è possibile creare delle librerie di programmi oggetto da utilizzare nei programmi al bisogno; dopo avere compilato il testo tramite il programma BC, viene creata la libreria indicata e viene aggiunto a quest'ultima il codice oggetto del programma compilato. È opzionalmente possibile compilare il testo con i dati utili al controllo degli errori run-time. Anche per questa possibilità di Quick Basic, si rimanda al Capitolo 4 per una più dettagliata spiegazione.

3.3 IL DEBUGGING

3.3.1 Esecuzione di istruzioni singole

Per potere rendersi conto di ciò che accade istruzione dopo istruzione in un programma che presenta degli errori logici (legati cioè ai risultati che lo stesso deve fornire), è possibile in Quick Basic l'esecuzione delle stesse una per una tramite il testo F8 (Single Step Mode). Se l'istruzione eseguita fornisce in qualche modo dei risultati sullo schermo dell'utente, quest'ultimo viene visualizzato per il tempo dell'esecuzione per poi tornare velocemente all'ambiente QB. Per controllare gli effetti dell'istruzione è sempre utilizzabile il tasto F4 che permette di visualizzare lo schermo utente in modo continuato. Per permettere l'esecuzione istruzione per istruzione ad iniziare dalla prima, basta scegliere l'opzione Restart del menu Run che porta l'esecuzione del programma alla prima linea.

3.3.2 Esecuzione di parti di programma e procedure singole

Per eseguire una parte qualsiasi di programma è invece sufficiente portare il cursore fino al punto in cui si vuole eseguire e premere il tasto F7; se il cursore non è visibile l'uso del tasto F7 corrisponde a quella del tasto F5 (Continue).

Al fine di eseguire una sola procedura è invece disponibile il tasto F10; se, invece di usare il tasto F8 quando è il momento di eseguire una Sub o una Function, si usa il tasto F10, questa viene eseguita interamente e i risultati forniti dalla stessa sono subito disponibili; l'esecuzione continua all'istruzione seguente la chiamata della Sub o della Function.
Nel caso in cui non si stesse per eseguire una Sub o una Function il tasto F10 agirebbe come F8.

3.3.3 Uso delle opzioni Trace ed History

È possibile, selezionando l'opzione **Trace On** (**Analizza il flusso**) nel menu Debug, attivare la modalità di **esecuzione animata**. Attivata la modalità suddetta, all'esecuzione del programma con Shift F5 o continuando ad eseguire lo stesso se precedentemente interrotto con Ctrl Break servendosi del tasto F5, Quick Basic farà in modo di eseguire ogni istruzione in maniera rallentata visualizzandola in maniera luminosa. Questa possibilità, usata in combinazione con tutte le altre date dal potente sistema di debug di Quick Basic, permette di controllare il flusso delle istruzioni di un programma sotto test, in maniera semplice ed efficace.

Per fermare l'esecuzione in modo animato, si devono utilizzare i tasti Ctrl Break; per disattivare questa modalità è necessario scegliere nuovamente l'opzione Trace On nel menu Debug; un segnale posto accanto all'opzione indicherà se la funzione è attiva o no;

Allo stesso modo dell'opzione Trace è possibile abilitare il funzionamento dell'opzione **History On** (**Resoconto**) del menu Debug; se tale operazione è attiva, le 20 precedenti istruzioni rispetto a quella in cui il programma è stato interrotto, vengono memorizzate per un successivo controllo. Infatti, tramite i tasti Shift F8, è possibile osservare, andando indietro nell'esecuzione, le 20 precedenti istruzioni eseguite mentre, con i tasti Shift F10 si va avanti. È una possibilità molto potente che offre Quick Basic di controllare il flusso delle istruzioni già eseguite da un programma; usata con gli altri strumenti di debug facilita enormemente la scoperta e l'eliminazione degli errori nei programmi.

Le due opzioni, Trace On ed History On, non influenzano il funzionamento dei tasti F7, F8 e F10.

3.3.4 I Breakpoints e l'opzione Set Next Statement

Un breakpoint è un punto del programma scelto dal programmatore in cui l'esecuzione dello stesso deve essere interrotta perché si sospetta che in quel punto esistano dei problemi; dopo l'interruzione, si possono controllare i contenuti di tutte le variabili per eliminare gli errori.

Per predisporre un breakpoint è sufficiente porre il cursore sulla linea in cui si vuole interrompere il programma e scegliere l'opzione **Toggle Breakpoint** (**Punto di interruzione**) del menu Debug (o premere semplicemente il tasto F9); per togliere un breakpoint, si agisce allo stesso modo; la linea con un breakpoint è visualizzata in reverse o in rosso. Per togliere tutti i breakpoints posti nel programma, senza dovere farlo uno per uno, si sceglie l'opzione **Clear All Breakpoints** (**Elimina ogni punto interruzione**) sempre nel menu Debug.

L'opzione **Set Next Statement** (**Imposta istruzione succ.**) del menu Debug cambia il flusso delle istruzioni eseguite dal programma in memoria. Dopo avere interrotto un programma è possibile farlo continuare in un altro punto diverso da quello in cui si è arrestato e per fare ciò si pone prima il cursore sulla linea da eseguire e poi si sceglie l'opzione suddetta che predispone Quick Basic ad eseguire l'istruzione indicata. Gli effetti di questa opzione sono paragonabili a quelli dell'istruzione GOTO e come per questa istruzione, non è possibile far continuare il programma dal mezzo di una procedura se questo si era interrotto a livello di un modulo.

3.3.5 La Watch Window ed i Watchpoints

Una finestra, chiamata Watch Window, viene aperta da Quick Basic quando si predispone la visualizzazione continua (monitoraggio) di una variabile o quando si predispone un watchpoint. Si può infatti, con l'opzione **Add Watch…** (**Osserva**) del menu Debug, predisporre la visualizzazione in tale finestra del contenuto delle variabili utilizzate dal programma e quando questo non è possibile l'indicazione <Not watchable> viene presentata accanto alla variabile interessata; infatti, per essere controllabili, le variabili devono essere visibili alla porzione di programma che si stava eseguendo al momento dell'interruzione.
Per ogni variabile elencata nella watch window vengono indicati i seguenti dati:

* il contesto

È indicato il nome del modulo nel cui contesto è visibile la variabile indicata; se il contesto attuale non è lo stesso, il contenuto della variabile viene indicato come <not watchable>.

* l'espressione o la variabile controllata

È poi visualizzata l'espressione o la variabile da controllare; è possibile introdurre sia il nome di una variabile (ad esempio, TOTALE) che una espressione completa (ad esempio, TOTALE*IVA/100).

Si possono, infine, predisporre delle espressioni logiche (**Watchpoints**) che vengono controllate in ogni momento durante l'esecuzione di un programma; se queste espressioni diventano vere, il programma viene interrotto. Ad esempio, tramite l'opzione **Watchpoint…** (**Punto di osservazione**) del menu Debug, si può introdurre l'espressione TOTALE>1000000 per interrompere il programma se il contenuto della variabile TOTALE diventa maggiore di 1000000; per i watchpoints vale quanto detto per il contesto nel controllo delle variabili e per quanto riguarda il risultato delle espressioni, questo è indicato come False o True a seconda delle situazioni nella Watch Window.

Per eliminare sia le variabili e le espressioni controllate che i watchpoints, si può agire in due modi distinti:

- si possono eliminare le singole voci aggiunte nella watch window tramite l'opzione **Delete Watch…** (**Elimina dall'osservazione**) del menu Debug;

- si possono eliminare tutte le voci della watch window usando l'opzione **Delete All Watch** (**Elimina ogni osservazione**).

Nel primo caso viene aperta una dialog box in cui si può scegliere con i tasti del cursore tra tutte le voci esistenti nella watch window al fine di eliminarne una in particolare.

La gestione della watch window è di particolare importanza al fine della scoperta ed eliminazione degli errori in un programma e, insieme a tutte le caratteristiche di debug del Quick Basic, fa dello stesso un linguaggio particolarmente potente e semplice allo stesso tempo.

3.3.6 Il menu Calls

Il menu **Calls** (**Chiamate**) è l'unico in Quick Basic in cui non compaiono comandi. Esso è infatti costituito dai nomi di tutte le procedure richiamate dall'inizio dell'esecuzione di un programma. Il nome più in alto nell'elenco è quello attivo al momento; per vedere i punti in cui sono state richiamate tutte le procedure elencate, basta portare il cursore nel menu Calls, sul nome della procedura chiamante per ottenere le linee di programma relative. È un metodo molto potente per il controllo del flusso di un programma in cui compaiano molte Subs e Functions. Il numero delle procedure controllate è limitato dall'ampiezza dello stack interno.

3.4 IL MENU DI AIUTO

3.4.1 Generalità sull'aiuto in linea

Se si dovesse, in qualsiasi momento, avere dei dubbi circa la sintassi di una istruzione o di una funzione, o non si ricordasse il modo di operare dei tasti di editing, il Quick Basic mette a disposizione delle pagine di aiuto richiamate molto semplicemente servendosi del menu **Help** (**Guida**) o di alcuni tasti adibiti allo scopo.

L'efficienza dell'help in linea, già molto grande nella versione 4.0 del Quick Basic, viene notevolmente aumentata nella versione 4.5 in cui si può affermare di avere un vero manuale del linguaggio in linea (v. 3.4.4).

3.4.2 Le pagine di aiuto generale

Usando l'opzione **General…** del menu Help o il tasto F1, vengono visualizzate le schermate di aiuto generale. Queste consistono di 3 videate il cui contenuto è il seguente:

*** Schermata 1**

È quella che contiene la maggior parte dei tasti di editing e di debugging usati dal Quick Basic con la relativa funzione eseguita. I tasti sono divisi in blocchi funzionali riguardanti l'**inserimento**, la **selezione**, la **cancellazione**, la **copia**, la **ricerca** del testo di un programma e il **debug** dello stesso.

*** Schermata 2**

Consiste nella prima parte della tabella ASCII quella relativa ai codici compresi tra 0 e 127. Vengono visualizzati, per ogni simbolo, il codice numerico, il simbolo visualizzato e per i caratteri speciali (da 0 a 31) il codice che li caratterizza.

*** Schermata 3**

È la seconda parte della tabella ASCII in cui compaiono i simboli con codice compreso tra 128 e 255. In questa parte sono inclusi i simboli semi-grafici che permettono di costruire le cornici in modo testo; ci sono poi i caratteri dell'alfabeto greco e, in generale, tutti quelli ammessi dallo standard IBM.

Per potersi spostare tra le 3 schermate suddette, si può usare il tasto Tab tra le indicazioni che compaiono nella parte inferiore di ogni videata. Premendo il tasto Return quando è attiva l'opzione **Next** si avanza una pagina mentre si torna alla pagina precedente con l'opzione **Previous**; è possibile utilizzare anche le iniziali (N e P) delle due opzioni per effettuare gli spostamenti tra le videate. Per abbandonare tali schermate si usa l'opzione **Cancel** o, come in tutte le schermate di help, basta premere il tasto Esc.

Selezionando l'opzione **Keywords** si accede ad una ulteriore schermata in cui vengono visualizzate le oltre 200 istruzioni e funzioni del linguaggio. Ne viene visualizzato, in ordine alfabetico, un primo gruppo e con i tasti di posizionamento del cursore, ci si può spostare per visualizzare le restanti. Selezionata un'istruzione o una funzione di questo elenco, viene visualizzata una pagina di aiuto relativa che è possibile eliminare con il tasto Esc. Queste sono le

pagine di aiuto relative alla sintassi del linguaggio che si possono ottenere anche grazie al modo context-sensitive.

3.4.3 Aiuto context-sensitive

Posizionando il cursore nel testo su un'istruzione o una funzione di Quick Basic e aprendo il menu di Help, tramite l'opzione **Topic** si accede direttamente alla pagina di aiuto interessata senza dovere passare dall'elenco completo delle keywords relative all'aiuto generale. In alternativa, dopo aver posizionato il cursore sull'istruzione di cui si vuole la schermata di aiuto, basta premere i tasti Shift F1 senza intervenire nel menu di Help.

3.4.4 Differenze con l'aiuto della versione 4.5 di QB

Il supporto di aiuto fornito dalla versione 4.5 di Quick Basic (versione italiana) è molto diverso e val la pena di essere commentato. A differenza della versione 4.0, in questa esiste una **guida**, una sorta di manuale sempre in linea richiamabile con l'opzione **Uso della guida** del menu Guida o con i tasti Shift F1 che prima erano utilizzati per l'aiuto context-sensitive. Quest'ultimo è realizzato tramite il tasto F1 (oppure con l'opzione **Argomento** del menu Guida) dopo avere posizionato il cursore nel testo sull'istruzione di cui si vuole l'aiuto. La guida del Quick Basic 4.5 prevede molti argomenti tra i quali ci si può spostare con il tasto Tab; questi argomenti sono chiamati "hyperlink" e non sono altro che dei riferimenti a delle pagine di spiegazione.

Scegliendo l'opzione **Panoramica** del menu Guida si potrà scegliere tra due categorie di argomenti di aiuto: *Uso di QuickBASIC* e *Linguaggio di programmazione BASIC*. Nel primo gruppo esistono tutti gli aiuti relativi all'uso dei tasti di editing e di debugging, a come usare il comando QB all'apertura, ai limiti della versione 4.5 e ad altri argomenti di carattere generale. Nel secondo gruppo vengono fornite informazioni più specifiche sul linguaggio, sui tipi dei dati disponibili, sulle espressioni e gli operatori, sulle istruzioni e le funzioni organizzate per gruppi, sui caratteri ASCII e sui codici della tastiera, e, inoltre, è fornito un nutrito gruppo di programmi di esempio divisi per argomento.

È poi disponibile l'opzione **Indice** del menu Guida con cui si può scegliere la visualizzazione dell'aiuto relativo a ogni parola chiave del linguaggio, parole che sono elencate in ordine alfabetico ed attraverso le quali si ci può spostare molto semplicemente. Per ogni parola chiave esistono degli esempi, anche approfonditi, dai quali potere estrarre delle idee per realizzare i propri programmi.

Questa versione è indubbiamente da preferire alla precedente per quanto riguarda l'organizzazione dell'aiuto (tutto in italiano) che è messo a disposizione del programmatore.

CAPITOLO 4

La compilazione

CAPITOLO 4

4.1 LA COMPILAZIONE

4.1.1 Il programma BC

Insieme al programma QB viene anche fornito il programma BC e cioè il **Basic command-line compiler**. Questo programma è utilizzato per compilare i programmi scritti in Quick Basic quando ci si trova all'esterno dell'ambiente QB e cioè in DOS. Questo programma viene richiamato automaticamente dall'ambiente QB quando si sceglie l'opzione Make EXE File del menu Run (v. 3.2.3) ma è preferibile richiamarlo da DOS, senza trovarsi nell'ambiente, per diversi motivi:

* uso di altri editors

È possibile scrivere i programmi in Quick Basic usando altri programmi adatti allo scopo (ad esempio Wordstar) per poi compilarli sotto DOS;

* problemi di memoria

Se si hanno problemi di memoria quando si compila un grosso programma con librerie molto grandi in ambiente QB, si può abbandonare quest'ultimo ed utilizzare da DOS il programma BC che ha così a disposizione una quantità maggiore di memoria;

* creazione di file oggetto

In alcuni casi si deve solo creare il file oggetto (.OBJ) di un programma che poi sarà collegato con altri e si deve quindi compilare il programma ma non eseguirlo; questo è possibile con il programma BC;

* compatibilità con il programma Code View

Se si deve creare un file compatibile con il programma di debug Code View, è necessario utilizzare il programma BC.

Per compilare un programma scritto in Quick Basic con il programma BC, si deve utilizzare lo stesso con la seguente sintassi:

BC [sourcefile[,[objectfile[,[listingfile]]] [optionlist][;]

oppure semplicemente **BC**.

Nel secondo caso tutti i parametri vengono dati in maniera interattiva dopo la richiesta del compilatore. La descrizione dei parametri richiesti è la seguente:

sourcefile È il nome del file sorgente che contiene il programma scritto in Quick Basic che si vuole compilare. Può essere qualsiasi con una qualunque estensione, ma se questa manca

viene assunta l'estensione standard .BAS. Se si vuole specificare un file senza estensione si deve far seguire un punto al suo nome.

objectfile È il nome del file oggetto creato dal compilatore BC. Contiene la traduzione in linguaggio macchina del programma sorgente ed è organizzato in maniera da essere collegabile ad altri programmi compilati con BC. Le regole per l'estensione sono le stesse di quelle adottate per il sourcefile tranne per il fatto che l'estensione standard prevista è .OBJ. Se questo nome non viene specificato, viene assunto quello del sourcefile con estensione .OBJ.

listingfile Questo file è quello in cui viene registrato il listato del programma sorgente completo di tutte le informazioni derivanti dalla compilazione inclusi gli errori compile-time. Le regole per la composizione del nome di tale file sono le stesse di quelle usate per l'objectfile ma l'estensione standard prevista è .LST.

optionslist È l'elenco delle opzioni utilizzabili per definire alcuni modi non standard di funzionamento del compilatore BC.

Le opzioni utilizzabili con il programma BC sono le seguenti:

/a Se il listingfile deve essere creato, con questa opzione pone anche il listato assembler del programma per ogni compilata.

/ah Permette che gli arrays dinamici occupino tutta la memoria disponibile. Senza tale opzione il limite di memoria utilizzabile dagli arrays dinamici è di 64 Kbytes.

/C:grandezza buffer Predispone la grandezza del buffer di ricezione per le comunicazioni seriali. Il valore di default è 512 ma questo può essere variato fino a 32767. Il buffer di trasmissione ha invece la lunghezza di 128 byte che non è modificabile in alcun modo. Tutto ciò ha effetto solo se è presente almeno una scheda di interfaccia seriale.

/d Genera il codice necessario al controllo degli errori run-time e permette di usare i tasti Ctrl Break per interrompere l'esecuzione del file EXE. È la stessa opzione usata dall'opzione Make EXE File del menu Run nell'ambiente QB.

/e Permette l'utilizzo dell'istruzione ON ERROR GOTO nel testo sorgente. È usata molte volte in unione all'opzione /x.

/mbf L'uso di questa opzione fa in modo che Quick Basic usi il formato Microsoft Binary al posto del formato IEEE nella conversione dei valori numerici. A tal fine le funzioni CVS e CVD vengono sostituite automaticamente dalle funzioni CVSMBF e CVDMBF e le funzioni MKS$ e MKD$ dalle funzione MKSMBF$ e MKDMBF$.

/o Sostituisce la libreria denominata BCOM40 con la libreria BRUN40 nella compilazione dei testi sorgenti. Questo permette al programma LINK di creare un file di tipo stand-alone, eseguibile cioè senza la necessità di avere il file BRUN40.EXE.

/r Registra gli arrays in ordine di riga invece che di colonna. Questo è necessario se si deve collegare il programma ad altri scritti in linguaggi che lo richiedano.

/s Registra le costanti alfanumeriche nel file oggetto invece che nella tavola dei simboli appositamente creata allo scopo. Questa opzione va usata se il messaggio di errore ‘Out of memory’ appare durante l’esecuzione di un programma che contiene numerose costanti alfanumeriche.

/v Abilita il controllo degli eventi relativi alle comunicazioni (COM), alla penna luminosa (PEN), al joystick (STRIG), al timer (TIMER), al buffer musicale (PLAY). Il controllo viene effettuato ad ogni istruzione.

/w Funziona allo stesso modo dell’opzione /v con la differenza che controlla gli eventi ad ogni linea del testo sorgente.

/x Permette di utilizzare le istruzioni RESUME, RESUME 0 e RESUME NEXT congiuntamente all’istruzione ON ERROR.

/zd Produce un file oggetto compatibile con il programma di debug SYMDEB della Microsoft fornito con il Macro Assembler v. 4.0.

/zi Produce un file oggetto compatibile con il programma di debug Code View della Microsoft fornito con il Macro Assembler v. 5.0 o con il compilatore C v. 6.0.

È possibile immettere in ogni nome di file coinvolto nella compilazione, un percorso adeguato, eventualmente completo di drive, da cui leggere o in cui scrivere i file adeguati. Ad esempio, se si volesse compilare il file PROVA.BAS che si trova nella sottodirectory QB45 della root e si volesse scrivere il file oggetto e il file di listato nella root, si dovrebbe dare la seguente linea di comando:

BC \QB45\PROVA, \, \;

4.1.2 Il programma LINK

Dopo aver compilato il testo sorgente bisogna collegare (to link) il file oggetto così generato a tutti gli altri file oggetto necessari e a tutte le librerie utili, per creare il file eseguibile dal DOS. Per fare questo lavoro è stato preparato il LINK, programma molto potente, senza il quale i programmi scritti con i linguaggi compilatori sarebbero inutili.
La sintassi di tale programma è la seguente:

LINK objectfile[,[exefile[,[mapfile[,[lib]]]] [linkopts][;]

oppure semplicemente **LINK** (tenere presente che nel primo caso la linea non può essere più lunga di 128 caratteri.

È anche possibile la forma **LINK @respfile** in cui il file indicato contiene tutte le risposte che dovrebbero essere fornite al programma LINK se questo fosse stato chiamato nella seconda maniera.

Infatti in questo caso tutti i parametri vengono dati in maniera interattiva dopo la richiesta del programma LINK.

La descrizione dei parametri richiesti è la seguente:

objectfile È il nome del file oggetto o dei file oggetto che si vogliono collegare tra loro e alle librerie. Se i file sono più di uno, devono essere separati dal simbolo + o dallo spazio. L'estensione .OBJ per tali file è di default. Ad esempio, se si volessero collegare i file oggetto PROVA.OBJ e PROVA2.OBJ si dovrebbe dare la seguente linea di comando:

LINK PROVA+PROVA2;

exefile È il nome dell'unico file eseguibile creato dall'unione dei file oggetto precedentemente trattati. L'estensione obbligatoriamente usata è EXE ma può pensare anche il programma LINK ad aggiungerla. Se il nome di questo file mancasse, sarebbe assegnato il nome del file oggetto con l'estensione EXE. Nel caso in cui ci fossero più file oggetto, sarebbe assegnato il nome del primo nella lista. Nell'esempio precedente il LINK predispone il nome PROVA.EXE al file eseguibile.

mapfile Questo file contiene un listato della mappa dei simboli utilizzati nel collegamento dei vari file oggetto. L'estensione di questo tipo di file è per default .MAP ed il nome se manca, è quello del primo file oggetto della lista. Al posto di questo nome si può usare un nome di periferica di MS-DOS tale da fornire le informazioni al posto di registrarle; per visualizzarle si utilizza la periferica CON, per stamparle la PRN e se non si vuole usare questa opzione, la periferica NUL.

lib È una lista di nomi di librerie in cui si trova parte del codice oggetto necessario alla realizzazione del file eseguibile. Si possono specificare fino a 16 librerie separate dal simbolo + o dallo spazio e l'estensione standard per tali file è .LIB. Le librerie standard usate dal linguaggio Quick Basic sono automaticamente utilizzate dal programma LINK. Ad esempio, nel caso in cui ci fossero delle routines contenute nella libreria BPLUS.LIB ed utilizzate dal programma PROVA, per collegare correttamente il file oggetto generato dal programma BC e la libreria BPLUS, si dovrebbe fornire la seguente linea di comando:

LINK PROVA,,NUL,BPLUS;

Notare che al posto del nome del mapfile, è stato scritto il nome della periferica NUL che permette di non generare alcun mapfile visto che questo non è richiesto. Se si fosse omessa tale indicazione, sarebbe stato creato il file PROVA.MAP contenente le informazioni riguardanti i simboli utilizzati durante il collegamento.

linkopts È l'elenco delle opzioni utilizzabili per definire alcuni modi non standard di funzionamento del programma LINK. Le opzioni utilizzabili con il programma LINK sono le seguenti:

/BA[TCH] Questa opzione fa in modo che il programma LINK non chieda un nuovo percorso di ricerca nel caso in cui una libreria non venga trovata; in questo caso continua senza usare tale file.

/CO[DEVIEW] L'opzione /CODEVIEW prepara il codice eseguibile ad essere utilizzato con il debugger Code View della Microsoft. Questo avviene solo il file sorgente è stato compilato con l'opzione /zi. Questa opzione è incompatibile con l'opzione /Q.

/E[XEPACK] Questa opzione permette di ottimizzare il codice del file eseguibile talmente da ridurlo in maniera drastica (molte volte anche più del 60%) eliminando alcune sequenze di caratteri inutili. Il caricamento e la partenza di un programma realizzato in tale modo è più veloce del normale.

/HE[LP] Con questa opzione vengono ignorati tutti i dati precedentemente descritti, e vengono solamente elencati su video tutte le opzioni utilizzabili con il comando LINK.

/I[NFORMATION] Con l'opzione /I vengono fornite alcune informazioni durante il processo di collegamento, utili a determinare le locazioni in cui i file oggetto vengono posti dal programma LINK nel file eseguibile.

/M[AP]:number Questa opzione permette la creazione di un file con estensione .MAP anche se non è stato specificato nella linea di comando, contenente tutte le informazioni relative ai simboli pubblici usati dal linker e dal compilatore nel programma utente. Per ogni segmento usato viene specificato il punto di inizio e di fine relativo all'inizio del modulo caricabile, la lunghezza, il nome e la classe di appartenenza. Per i simboli invece soltanto l'indirizzo e il nome; essi sono elencati in ordine alfabetico se il loro numero non eccede 2048; in questo caso questi non sono elencati in ordine a meno che nell'opzione MAP non venga specificato un numero superiore a 2048 tale da permetterne l'ordinamento. Tutti i simboli il cui nome inizia per B$ o termina per QQ sono riservati al compilatore o al linker. I simboli che hanno prima del nome la specifica Abs, sono valori assoluti e non indirizzi di memoria.

/NOD[EFAULTLIB…] Normalmente il compilatore BC include nel file oggetto il nome della libreria standard che il programma LINK deve utilizzare anche se non specificata nel parametro 'lib' di quest'ultimo (normalmente BCOM40.LIB o BRUN40.LIB). Specificando questa opzione non viene usata tale libreria standard e a questo punto bisogna specificare il percorso ed il nome della libreria usata.

/NOI[GNORECASE] Questa opzione permette al programma LINK di distinguere tra le lettere maiuscole e quelle minuscole. Non si deve usare quando si utilizza il LINK per i programmi scritti in Quick Basic.

/NOP[ACKCODE] Questa opzione è normalmente usata dal programma LINK per disabilitare la combinazione di segmenti di codice adiacenti in gruppi; è intesa per default. Se, per esempio, la variabile d'ambiente LINK dovesse abilitare il 'segment-code packing' tale opzione la disabiliterebbe.

/PACKC[ODE]:number Questa opzione viene usata per combinare i segmenti di codice adiacenti. Il numero fornito è il massimo numero di segmenti di codice adiacenti riuniti in un gruppo dal linker prima di formarne un altro; il valore di default è 65530.

/PACKD[ATA]:number Vale quanto detto per la precedente opzione, ma per quanto riguarda i segmenti di dati adiacenti.

/PAU[SE] Questa opzione fa in modo che il programma LINK si arresti prima di scrivere il file .EXE su disco per permettere all'operatore di cambiare quest'ultimo se ce ne fosse la necessità. Naturalmente non è usata se si opera con il disco fisso.

/Q[UICKLIB] È l'opzione usata per la creazione delle Quick Libraries; si può fare ciò anche dall'ambiente QB tramite l'opzione Make Libraries… del menu Run. Non bisogna dimenticare di fornire al programma LINK il nome della libreria di supporto denominata BQLB40.LIB. Con questa opzione non è possibile usare l'opzione /EXEPACK. Per dei maggiori chiarimenti circa le Quick Libraries, si rimanda alla sezione 4.2.

/SE[GMENTS]:number Predispone il LINK ad usare un certo numero di segmenti che può andare da 1 a 1024. Normalmente vengono usati 128 segmenti. L'uso di ogni segmento fa in modo che venga usata più memoria dal programma LINK all'atto del collegamento; se questa memoria non è disponibile il LINK emette un messaggio d'errore. Un messaggio d'errore viene altresì fornito se viene specificato un numero di segmenti minore di quelli necessari all'operazione di collegamento in corso.

/ST[ACK]:number Con questa opzione si specifica la grandezza dello stack per il programma utente. Il massimo valore è 65535 e lo standard imposto dalle librerie del Quick Basic è 2048.

Altre opzioni disponibili con il programma LINK non sono utilizzate per i programmi compilati con BC o, comunque, servono a compiere azioni che il Quick Basic effettua automaticamente.

Per quanto riguarda le librerie, il LINK utilizza le librerie standard il cui nome è registrato nei file oggetto all'atto della creazione degli stessi. Queste librerie, come quelle non standard, vengono ricercate all'interno della directory di lavoro; possono essere specificati altri percorsi per le librerie non standard immettendoli dopo la terza virgola nella linea di comando del LINK o dopo la richiesta esplicita dello stesso; inoltre le librerie possono essere trovate dal LINK utilizzando la variabile di ambiente LIB. Utilizzando l'opzione /NODEFAULTLIBRARYSEARCH è possibile non utilizzare la libreria standard imposta dal file oggetto ed inserire la libreria usata nella linea di comando completa di percorso.

Durante il processo di collegamento il programma LINK utilizza tutta la memoria disponibile. Se questa non dovesse bastare, nello scrivere il file eseguibile il LINK si servirebbe di un file temporaneo utilizzando quindi il disco come memoria. Il nome di questo file varia a seconda della versione di DOS usato; prima della versione 3.0 il file temporaneo si chiama VM.TMP mentre per le versioni di DOS che vanno dalla 3.0 in poi, il nome del file temporaneo viene assegnato dal DOS. La directory in cui viene creato tale file è quella indicata dalla variabile di ambiente TMP ed in mancanza di questa, viene utilizzata la directory di lavoro.

Solo quando si lavora con i dischetti ed il file temporaneo viene creato su quest'ultimi, il LINK emette un messaggio che avverte di non estrarre il disco in cui il file è stato creato prima che il processo termini. Se questa operazione viene effettuata, non si può essere sicuri degli effetti prodotti sul file eseguibile.

È possibile utilizzare una variabile di ambiente chiamata LINK per memorizzare alcune opzioni più comunemente usate con il programma LINK. Queste saranno usate automaticamente ogni volta che il programma LINK verrà utilizzato. Le opzioni usate nella linea di comando sono comunque prevalenti rispetto a quelle usate nella variabile LINK. Ad esempio, se si volesse che tutti i programmi eseguibili avessero il codice per il programma Code View, basterebbe prima preparare la variabile d'ambiente LINK nel seguente modo

SET LINK=/CO

si possono poi realizzare tutti i file eseguibili dei programmi interessati con la semplice linea:

LINK nomeprogramma;

per ottenere i file .EXE con il codice per il Code View.

4.2 GESTIONE DELLE LIBRERIE

4.2.1 Il concetto di libreria

Nella realizzazione dei programmi si utilizzano normalmente dei sottoprogrammi o delle funzioni che ne facilitano la composizione. Alcune funzioni e sottoprogrammi sono inclusi nel linguaggio usato (ad esempio, nel caso del Quick Basic, LOG(n) per i logaritmi naturali e SQR(n) per la radice quadrata) ed il codice oggetto relativo è incluso nei file chiamati **librerie standard**. Sono dei file, come il BCOM40.LIB, che contengono il codice oggetto di un insieme di funzioni e sottoprogrammi che, in definitiva, costituiscono il linguaggio Quick Basic. Il codice oggetto di tali sottoprogrammi e funzioni deriva da dei programmi scritti a loro volta in linguaggi di più basso livello a loro volta trasformati dai compilatori adeguati. È il compilatore, sia quello che agisce se ci si trova nell'ambiente QB sia il programma BC, che per ogni istruzioe e funzione del nostro programma sorgente, compone le chiamate a tutte queste porzioni di codice oggetto per realizzare quello che sarà poi il file con estensione .OBJ. In seguito il programma LINK provvederà a ricercare, estrarre e comporre tutte le parti di codice oggetto presenti nelle librerie di cui esiste un riferimento nel file oggetto per la creazione di un file eseguibile (con estensione EXE).

Ma è anche possibile scrivere in Quick Basic un sottoprogramma o una funzione di uso generico, quale può essere quella che calcola il logaritmo di un numero in base 10, e creare una **libreria non standard**. Sarà poi possibile chiamare in un programma utente tale funzione, compilare lo stesso ed eseguire il programma LINK avendo cura di specificare che il contenuto di tale libreria sara' utilizzata dal nostro programma.

4.2.2 Le librerie .LIB

Le librerie utilizzate dal programma LINK sono quei file aventi estensione .LIB. Il contenuto di tali file risponde a delle regole ben precise per far sì che il programma LINK possa funzionare correttamente con esse. Le porzioni di codice oggetto in esse contenute possono essere gestite tramite il programma LIB della Microsoft (v. 4.2.4). Le librerie standard fornite con il Quick Basic sono:

BCOM40.LIB utilizzata nella compilazione normale dei file sorgenti scritti in Quick Basic;

BRUN40.LIB utilizzata nella compilazione dei file sorgenti scritti in Quick Basic se si usa l'opzione /o;

BQLB40.LIB utilizzata dal programma LINK per la realizzazione delle librerie .QLB;

QB.LIB contenente alcuni sottoprogrammi particolari di Quick Basic che permettono l'interfaccia a programmi scritti in Assembler.

4.2.3 Le librerie .QLB

Le Quick Libraries sono concettualmente identiche alle librerie di tipo .LIB con la differenza che esse sono usate dal programma QB per permettere la compilazione, e quindi l'esecuzione, di quei programmi che utilizzano funzioni o sottoprogrammi non standard il cui codice oggetto è contenuto nelle librerie .LIB. È infatti da quest'ultime che esse vengono costituite e sono necessarie solo per le funzioni e i sottoprogrammi non standard; infatti tutte le istruzioni e funzioni normalmente usate da Quick Basic, vengono regolarmente compilate ed eseguite nell'ambiente QB senza l'aiuto di alcuna libreria di tipo .QLB. Per permettere al programma QB di avvalersi del contenuto di tali librerie non standard all'interno dell'ambiente, le Quick Libraries vengono caricate all'atto della chiamata del programma QB facendo seguire il loro nome all'opzione /L. Ad esempio, per caricare in memoria il contenuto della libreria non standard BPLUS.QLB derivante dalla libreria BPLUS.LIB, si deve fornire la seguente linea di comando:

QB /LBPLUS

Per creare le librerie di tipo .QLB da quelle di tipo .LIB si utilizza l'opzione /Q del programma LINK tenendo presente che il programma suddetto deve avere a disposizione il contenuto di un'altra libreria di tipo .LIB, la BQLB40.LIB, atta allo scopo. Per creare, ad esempio, la libreria BPLUS.QLB dalla libreria BPLUS.LIB si darà il seguente comando:

LINK BPLUS.LIB,BPLUS.QLB,NUL,BQLB40.LIB /Q

4.2.4 Il gestore delle librerie Microsoft LIB

Visto che le librerie sono formate da pezzi di codice oggetto provenienti dai programmi originali scritti in linguaggi di basso livello e compilati, si è reso necessario uno strumento atto a gestire il collegamento di questi in un file unico: tale strumento è il programma gestore di librerie della Microsoft denominato LIB.

Le potenzialità del programma LIB sono le seguenti:

- si possono combinare dei moduli oggetto per creare un file di libreria non esistente in precedenza;

- si possono aggiungere moduli oggetto ad un file di libreria già esistente;

- si possono rimpiazzare dei moduli oggetto contenuti in un file di libreria già esistente con i moduli aggiornati;

- si può cancellare un modulo oggetto non più utilizzato da una libreria già esistente;

- si possono estrarre i moduli oggetto da una libreria esistente e porli in un file oggetto separato;

- si possono combinare due o più librerie in una sola.

Nel caso in cui una libreria venisse aggiornata, il programma LIB agirebbe su una copia della libreria, lasciando per sicurezza l'originale intatto.

Per eseguire il programma LIB si utilizza la seguente linea di comando:

LIB oldlib[/**P**[AGESIZE]:num][**commands**][,[**listfile**][,[**newlib**]]][;]

oppure semplicemente **LIB** (tenere presente che nel primo caso la linea non può essere più lunga di 128 caratteri).

È anche possibile la forma **LIB @respfile** in cui il file indicato contiene tutte le risposte che dovrebbero essere fornite al programma LIB se questo fosse stato chiamato nella seconda maniera.

Solo in questo caso tutti i parametri vengono dati in maniera interattiva dopo la richiesta del programma LIB.

La descrizione dei parametri richiesti è la seguente:

oldlib È il nome della libreria di cui si vuole cambiare il contenuto o che si vuole creare. Nel secondo caso viene chiesta una conferma per la creazione.

/P:number Il numero fornito dopo tale opzione indica la grandezza di una pagina della libreria. Tutti i moduli sono allineati all'inizio di una pagina e quindi occupano uno spazio che è sempre multiplo di questa grandezza. La pagina può essere grande da 16 a 32768 includendo solo quei valori che sono potenze di 2; lo standard è 16.

commands Sono i comandi seguiti dai nomi dei moduli oggetto necessari alla gestione della libreria. Sono elencati in seguito.

listfile È il nome di un file in cui viene elencato il contenuto della libreria, modulo per modulo, con il nome, la posizione e la lunghezza degli stessi all'interno della libreria. Può essere il nome delle periferiche del DOS (CON o PRN) se si deve solo visualizzare o stampare. Non viene utilizzato se non ne è specificato alcuno.

newlib È il nome della nuova libreria creata dato che l'originale non viene modificata. Se non viene specificato, la vecchia libreria assume l'estensione .BAK e la nuova l'estensione .LIB.

I comandi utilizzati per la gestione della libreria (commands) sono i seguenti:

[+]objfile viene aggiunto il modulo indicato alla fine della libreria. Il simbolo + non è obbligatorio.

[+]libfile viene aggiunto il contenuto della libreria indicata alla fine della libreria modificata. Il simbolo + non è obbligatorio.

-module viene rimosso il modulo indicato dalla libreria. Il nome del modulo non deve avere il percorso o l'estensione.

-+module viene sostituito il modulo indicato con il nuovo che ha lo stesso nome. È una opzione utile all'aggiornamento dei moduli nelle librerie.

*module il modulo specificato viene copiato con il suo nome ed estensione .OBJ in un file contenuto nella directory di lavoro del drive corrente. Il modulo viene conservato nella libreria.

-*modulo come nel caso precedente, ma il modulo viene eliminato dalla libreria.

Ad esempio, se si volessero effettuare le seguenti operazioni sulla libreria PROVA.LIB

- aggiunta dei moduli MOD1, MOD2, MOD3
- rimozione del modulo MOD0
- aggiornamento del modulo MOD9

si dovrebbe fornire la seguente linea di comando

LIB PROVA MOD1 MOD2 MOD3 -MOD0 -+MOD9

Una libreria può raggiungere una grandezza che dipende dalla dimensione della sua pagina.

4.2.5 Esempio di realizzazione di una libreria

Si può, per esempio, realizzare una libreria che contenga un modulo in cui esistano le 2 funzioni non standard che calcolano il logaritmo in base 10 e la radice cubica di un numero.
Per potere scrivere in Quick Basic questo modulo si deve operare nel seguente modo:

* creare un Main Module contenente soltanto alcune righe di commento al programma chiamato FUNZSPEC;

* creare all'interno del modulo una Function chiamata LOGD contenente le istruzioni relative al calcolo del logaritmo in base 10 di un numero;

* creare all'interno del modulo una Function chiamata RAD3 contenente le istruzioni relative al calcolo della radice cubica di un numero.

Il programma è così fatto:

```
'(Main module FUNZSPEC.BAS)
DECLARE FUNCTION LOGD! (N!)
DECLARE FUNCTION RAD3! (N!)
'
'
' File contenente funzioni
'     non standard
'
'
'(Function LOGD)
FUNCTION LOGD(N) STATIC
    LOGD=LOG(N)/LOG(10)
END FUNCTION
'(Function RAD3)
FUNCTION RAD3(N) STATIC
    RAD3=N^(1/3)
END FUNCTION
```

Il file viene compilato a questo punto con il compilatore BC servendosi della seguente linea di comando:

```
BC FUNZSPEC;
```

viene quindi creato un file FUNZSPEC.OBJ contenente il codice oggetto del modulo contenente le funzioni non standard. A questo punto si può creare il file di libreria grazie alla linea di comando:

```
LIB FUNZSPEC +FUNZSPEC;
```

La nuova libreria viene creata ed è pronta per essere utilizzata. Da questa è possibile creare la libreria di tipo .QLB con il comando:

```
LINK FUNZSPEC.LIB,FUNZSPEC.QLB,NUL,BQLB40.LIB  /Q;
```

Si può adesso scrivere un programma che usi tali librerie, ad esempio il seguente:

```
'(Main module PROVA.BAS)
DECLARE FUNCTION LOGD! (N!)
DECLARE FUNCTION RAD3! (N!)
CLS
FOR T=1 TO 20
   PRINT T,LOGD(T),RAD3(T)
NEXT T
END
```

Il programma suddetto, dopo essere stato compilato con il comando

```
BC PROVA;
```

viene collegato alla libreria corrispondente tramite il comando

LINK PROVA,,NUL,FUNZSPEC;

che crea il file PROVA.EXE eseguibile sotto DOS.
Per eseguire il programma in ambiente QB è sufficiente invece il comando

QB /RUNPROVA /LFUNZSPEC

che carica la Quick Library FUNZSPEC.QLB in memoria, carica il programma PROVA.BAS e lo esegue.

CAPITOLO 5

Il linguaggio di programmazione

CAPITOLO 5

5.1 IL LINGUAGGIO DI PROGRAMMAZIONE

5.1.1 Le istruzioni e i comandi di Quick Basic

In questa parte del testo vengono elencate le istruzioni e i comandi utilizzabili in Quick Basic, in ordine alfabetico, completi di sintassi ed esempi di utilizzazione.

*** ACCESS**

Vedere OPEN

*** ALIAS**

Vedere DECLARE

*** AND**

L'operatore logico AND esegue una operazione tra due valori interi o interi lunghi, bit per bit, secondo le regole della tavola seguente:

1^ Operando	2^ Operando	Risultato
0 (FALSE)	0 (FALSE)	0 (FALSE)
0 (FALSE)	**1 (TRUE)**	0 (FALSE)
1 (TRUE)	0 (FALSE)	0 (FALSE)
1 (TRUE)	**1 (TRUE)**	**1 (TRUE)**

L'operatore può essere usato in qualsiasi espressione logico-aritmetica ma possiede il più basso livello di priorità tra gli operatori.

Esempi:

```
CLS
INPUT A,B
IF A>4 AND B<9 THEN
        PRINT A*B
ELSE
        PRINT A+B
END IF
```

Da notare in questo caso che l'operatore di AND logico avviene tra i valori (A>4) e (B<9) che possono assumere solo i due risultati FALSE o TRUE.

```
CLS
DEF SEG = &H40
POKE &H17, PEEK(&H17) AND &HBF
DEF SEG
```

In questo esempio l'operatore AND è usato per azzerare il bit numero 6 all'interno della locazione 0040:0017 in cui il DOS conserva lo stato del Caps Lock; in seguito a tale operazione, il modo di scrittura delle lettere diventa minuscolo e la corrispondente spia nella tastiera, se presente, viene spenta.
L'operazione effettuata è la seguente:

Locazione 0040:0017	=	x X x x x x x x
Valore &HBF	=	1 0 1 1 1 1 1 1
Risultato in 0040:0017	=	x 0 x x x x x x

*** ANY**
Vedere DECLARE

*** APPEND**
Vedere OPEN

*** AS**
Vedere COMMON, DECLARE, DEF, DIM, FIELD, FUNCTION, NAME, OPEN, REDIM, SHARED, STATIC, SUB, TYPE

*** BASE**
Vedere OPTION BASE

*** BEEP**

È l'istruzione usata per emettere un suono breve dall'altoparlante di cui è dotato il sistema, tale da avvertire l'utente del fatto che, per esempio, è occorso un errore durante l'esecuzione di un programma, o più semplicemente, che è terminata una parte di quest'ultimo. L'istruzione non ha alcun parametro e viene usata quindi, in questo modo:

BEEP

Si ottengono gli stessi effetti con la riga:

```
PRINT CHR$(7);
```

che emette il suono suddetto tramite il carattere <Bell> disponibile nella tabella ASCII (codice n. 7); è importante, se si usa questa forma, non dimenticare il simbolo ; che evita lo spostamento del cursore alla prossima riga.
Molte stampanti possono attivare il loro cicalino se viene loro mandato lo stesso codice con la riga seguente:

```
LPRINT CHR$(7);
```

*** BINARY**
Vedere OPEN

*** BLOAD**

Tramite questa istruzione è possibile caricare in memoria parti dello stesso precedentemente registrati su disco usando l'istruzione BSAVE. È un'istruzione presente anche in altre versioni del linguaggio Basic, ma è poco sfruttata dai programmatori che non ne hanno, forse, intuito le potenzialità.
È infatti utilissima, in coppia con BSAVE descritta in seguito, nel caso in cui si voglia gestire il video in maniera diversa dal normale; infatti nei programmi le maschere vengono, di norma, generate con delle istruzioni PRINT, rendendo scomoda, poco veloce e comunque molto pesante la parte degli stessi che si occupano della visualizzazione delle diverse videate che devono essere utilizzate. Con l'istruzione BLOAD è invece possibile caricare le maschere direttamente in memoria video e visualizzarle così molto velocemente con una sola istruzione; per fare ciò è sufficiente caricare il file che contiene le informazioni relative alla maschera, precedentemente registrate con l'istruzione BSAVE.

Ad esempio, la videata salvata su disco nell'esempio fornito nella descrizione dell'istruzione BSAVE, viene immediatamente riportata in memoria video con l'istruzione BLOAD, considerando il tipo di scheda video disponibile. È infatti noto che le schede video hanno una loro memoria riservata alla visualizzazione delle immagini e che questa ha indirizzi diversi a seconda del tipo di scheda; più precisamente, il **segmento** dal quale inizia la memoria varia seguendo le regole della prossima tabella:

Tipo Scheda (o modo video)	Segmento (esad.)
MDA, Hercules	B000
CGA, EGA, VGA	B800

Il punto del segmento dal quale inizia la memoria video (offset) è sempre 0 e quindi è sufficiente predisporre il valore del segmento prima dell'uso dell'istruzione BLOAD, per caricare correttamente il file in memoria; ad esempio, se si volesse caricare la schermata dell'esempio, usando una scheda monocromatica, si dovrebbe usare il seguente programma:

```
DEF SEG=&HB000              ' Predispone il segmento da usare
BLOAD "ESEMPIO.SCR",0       ' Carica il file ESEMPIO.SCR
DEF SEG                     ' Ritorna ad usare il segmento di default
```

La sintassi dell'istruzione prevede, oltre al nome del file da caricare, il valore dell'offset dal quale iniziare a caricare la memoria, ed avendo messo il valore 0 e predisposto il segmento B000, il contenuto del file ESEMPIO.SCR viene caricato ad iniziare dall'indirizzo B000:0000 destinato alla scheda video monocromatica, permettendo la visualizzazione di quello che prima era stato registrato con l'istruzione BSAVE. Nel caso in cui tale offset non venisse specificato, per default, verrebbero assunti i valori dell'offset e del segmento usati al momento della registrazione con BSAVE, dato che tali valori vengono registrati all'interno del file; qualunque altra definizione di segmento, in questo caso, verrebbe ignorata.

È inoltre possibile caricare in memoria direttamente il contenuto di un array precedentemente salvato con l'istruzione BSAVE. Questo potrebbe rendersi utile in due casi:

1. Non si ha spazio a disposizione in memoria centrale
2. Si devono utilizzare array di costanti

Nel primo caso è possibile fare spazio scaricando su disco gli arrays non necessari al momento per liberare memoria, a condizione che questi siano di tipo **dinamico**, come nell'esempio che segue:

```
' $DYNAMIC
DEFINT A-Z
CLS
DIM A(1 TO 1000)
FOR T=1 TO 1000
        A(T)=T
NEXT T
PRINT FRE(-1)
DEF SEG = VARSEG(A(1))
BSAVE "ARRAY.A%", VARPTR(A(1)),2000
DEF SEG
ERASE A
PRINT FRE(-1)
'…
'… Si è liberata della memoria da utilizzare
'…
DIM A(1 TO 1000)
DEF SEG = VARSEG(A(1))
BLOAD "ARRAY.A%", VARPTR(A(1))
DEF SEG
PRINT FRE(-1)
FOR T=1 TO 1000
        PRINT A(T);
NEXT T
END
```

Nell'esempio precedente un array dinamico di interi composto da 1000 elementi viene registrato su disco per liberare memoria, e viene in seguito caricato al suo posto nel momento in cui si rendesse necessario. L'array A è dichiarato esplicitamente dinamico dalla metaistruzione $DYNAMIC e viene salvato dall'istruzione BSAVE informando quest'ultima che lo stesso è memorizzato a partire dalla locazione puntata dal segmento ricavato con la funzione VARSEG e all'offset ricavato dalla funzione VARPTR; la lunghezza dell'array è determinata dal numero degli elementi per il numero dei bytes occupati da ogni elemento (2*1000 array intero, 4*1000 array singola prec., 8*1000 array doppia prec.); non è possibile usare questo metodo con gli arrays di stringhe. Dopo avere eliminato l'array (ERASE A) è disponibile la memoria da questo usato per altri scopi; l'array viene ricaricato in memoria dopo averlo ridimensionato e letto dal disco con l'istruzione BLOAD.

Nel secondo caso si caricherebbe da file un array, non importa se dinamico o statico, costante, preparato in precedenza, al posto di crearlo con delle frasi DATA; è infatti quest'ultimo un metodo molto dispendioso sia dal punto di vista dell'occupazione che della velocità dei programmi.

I nomi dei file usati dall'istruzione BLOAD (e BSAVE) hanno, per default, l'estensione BAS, ma è possibile, anzi molto consigliabile, usarne un'altra. Bisogna, inoltre, fare molta attenzione nel caricare in memoria dei file con tale istruzione dato che non viene fatto alcun controllo sulla destinazione in memoria e che quindi potrebbe, per errore, essere interessata memoria usata dal Quick Basic o dal DOS stesso, creando effetti indesiderati e imprevedibili.

*** BSAVE**

L'istruzione BSAVE è utilizzata per registrare su disco il contenuto di porzioni di memoria che poi possono essere ricaricate con l'istruzione BLOAD.
La sintassi dell'istruzione è la seguente:

BSAVE nomefile, offset, lunghezza

Il nomefile è il nome che avrà il file che si vuole creare e tale nome segue le regole evidenziate nella descrizione dell'istruzione BLOAD. L'offset è il valore dal quale inizia il salvataggio della memoria, e può essere 0 nel caso in cui si voglia registrare il contenuto dello schermo, determinato dalla funzione VARPTR nel caso in cui si voglia salvare il contenuto di un array; il segmento a cui si riferisce è quello di default del Quick Basic o quello determinato dall'istruzione DEF SEG; nel caso del video può essere B000 (scheda video monocromatica) o B800 (scheda video a colori) e per salvare il contenuto di arrays è determinato dalla funzione VARSEG.

Se si fosse in possesso di una scheda video monocromatica e si volesse registrare una schermata, si dovrebbe agire come nel seguente esempio:

```
CLS
K$=STRING$(80,196)
PRINT K$
PRINT TAB(38); "MENU"
PRINT K$
PRINT K$
LOCATE 23
PRINT K$
DEF SEG=&HB000
BSAVE "ESEMPIO.SCR",0,4000
DEF SEG
```

La schermata viene registrata sul disco nel file ESEMPIO.SCR ed occupa 4000 bytes (2000 caratteri sul video + 2000 attributi o colori) e 7 bytes che compongono la testata di un file del genere e che contengono i seguenti dati:

Numero byte	Funzione	Contenuto di ESEMPIO.SCR
1	Tipo file (BLOAD/BSAVE)	FD
2,3	Segmento di origine (L,H)	00, B0
4,5	Offset di origine (L,H)	00, 00
6,7	Lunghezza in bytes (L,H)	A0, 0F

in cui B000 è il segmento della scheda monocromatica, 0000 è l'offset da cui si è iniziato a salvare, 0FA0 è la lunghezza in bytes della zona salvata (4000 bytes).

Nel caso in cui si registrino degli arrays, questi valori sono molto variabili, dato che il segmento e l'offset dell'array dipendono dalla situazione interna della memoria usata da Quick Basic in un determinato istante, ed il programma tipo per compiere tale operazione è il seguente:

```
DIM A%(1 TO 100)
DIM B!(1 TO 100)
DIM C#(1 TO 100)
DEF SEG=VARSEG(A%(1))
BSAVE "ARRAY.A%", VARPTR(A%(1)), 200
DEF SEG=VARSEG(B!(1))
BSAVE "ARRAY.B!", VARPTR(B!(1)), 400
DEF SEG=VARSEG(C# (1))
BSAVE "ARRAY. C#", VARPTR(C# (1)), 800
DEF SEG
```

Notare come le funzioni VARSEG e VARPTR siano utilizzate per ricavare il segmento e l'offset del primo elemento dell'array dato che tutti gli altri sono disposti di seguito in memoria, e come la lunghezza in bytes vari a seconda del tipo di array.
Non è possibile salvare il contenuto di arrays stringa.

*** BYVAL**

Vedere CALL, CALLS, DECLARE

*** CALL**

Questa istruzione è usata per richiamare una routine, sia che questa sia scritta in Quick Basic che in un altro linguaggio. Nel primo caso ha la seguente sintassi

CALL nomeprocedura (par1, par2, …)

Il nome della procedura (definita come SUB appartenente al modulo che la sta richiamando) può essere lungo fino a 40 caratteri ed a questa possono essere passati diversi parametri da elaborare. Il passaggio di questi dati, in questo caso, avviene per **near reference**, viene cioè passato l'indirizzo senza il segmento in cui si trova il valore utilizzato e, nel caso in cui venga passato il contenuto di una variabile, questo può essere variato dalla SUB stessa. Ad esempio, dal seguente programma:

```
'(Main Module)
CLS
A=5
PRINT "IN MAIN MOD. IL VALORE DI A E' "; A
CALL AUMENTA(A)
PRINT "IN MAIN MOD. IL VALORE DI A E' "; A
END
'(Sub AUMENTA)
SUB AUMENTA (VAR) STATIC
```

```
        PRINT “NELLA SUB IL VALORE DI A E’ “; VAR
        VAR=VAR+5
END SUB
```

saranno stampati i valori 5, 5 e 10 a conferma del fatto che il contenuto della variabile A viene modificato nella Sub nel seguente modo:

- la variabile A assume valore 5 nel Main Module

- l’indirizzo della variabile A viene passato alla Sub AUMENTA, la quale si riferirà a quest’ultimo usando la variabile locale VAR appositamente creata; non a caso quest’ultima è dello stesso tipo di A (singola precisione) ed il suo valore è quello di A;

- viene aumentato il contenuto di VAR portandolo a 10 e tale valore viene posto in memoria nell’indirizzo in cui era contenuta la variabile A;

- tornando al Main Module, la variabile locale VAR viene eliminata, ma il valore di A è diventato 10 anche nel Main Module.

È interessante notare i seguenti fatti:

- la variabile A e la variabile VAR hanno diverso nome ma puntano allo stesso dato; nel caso in cui fossero presenti più parametri è quindi importante l’ordine con cui essi vengono passati dal modulo alla Sub;

- nella Sub non è possibile usare una variabile con nome VAR di tipo statico; gli argomenti sono infatti solamente locali costituendo solo un nome con cui riferirsi a dati già esistenti e non possono quindi essere definiti come autonomi;

- nel caso in cui venga usata una variabile nel modulo principale con nome uguale a quello del parametro della Sub e che questa venga condivisa tramite l’istruzione SHARED, all’interno della Sub essa non viene considerata e la variabile locale ha il sopravvento.

Le stesse considerazioni possono essere fatte nel caso in cui debbano essere passati come parametri degli arrays. In questo caso, ad esempio, si agirà come nel seguente esempio:

```
‘(Main Module)
CLS
DIM AR(1 to 100)
FOR T=1 TO 100
      AR(T)=T
NEXT T
CALL TRASFORMA(AR())
FOR T=1 TO 100
      PRINT AR(T);
NEXT T
END
‘(Sub TRASFORMA)
```

```
SUB TRASFORMA(X())
      FOR T=LBOUND(X) TO UBOUND(X)
            X(T)=SQR(X(T))
      NEXT T
END SUB
```

In questo programma di prova, il valore degli elementi di un array viene trasformato nella loro radice quadrata; da notare che nell'istruzione CALL il parametro rappresentato da un array viene passato con il nome seguito dalle parentesi aperte e chiuse e che nella Sub le funzioni LBOUND e UBOUND servono ad identificare i valori estremi del dimensionamento dello stesso array. L'array X è di tipo locale e si comporta esattamente come nel caso delle variabili semplici.

È possibile omettere la parola CALL e chiamare direttamente la Sub con il suo nome, ma la stessa deve essere dichiarata in precedenza con una frase DECLARE e bisogna omettere le parentesi nella dichiarazione dei parametri; nell'esempio precedente, le modifiche da apportare sono le seguenti:

```
'(Main Module)
DECLARE SUB TRASFORMA(X())
...
TRASFORMA  AR()
…
END
…
END SUB
```

Alcune volte è necessario che il valore di variabili o elementi di array non debba essere modificato da una Sub chiamata servendosi dell'istruzione CALL; in questo caso si deve agire, per una variabile semplice, in questo modo:

```
CALL AUMENTA ((A))
```

Le parentesi che racchiudono la variabile A, fanno in modo che questa venga considerata come un'espressione e quindi il suo valore viene copiato in un'area temporanea il cui indirizzo viene passato alla Sub; il riferimento sarà quindi di una copia della variabile e si potrà variarlo senza modificare il valore di quest'ultima.
Nel caso dell'array si deve usare un array locale temporaneo su cui agire per non modificare quello usato dal Main Module; ecco di seguito il codice per realizzare tale operazione:

```
'(Sub PROVA)
SUB PROVA (X())
      LB=LBOUND(X)
      UB=UBOUND(X)
      DIM Y(LB TO UB)
FOR EL=LB TO UB
      Y(EL)=X(EL)
NEXT EL
…
```

```
END SUB
```

Se si dovesse chiamare una routine scritta in altri linguaggi, come ad esempio C, si potrebbero rendere utili altri due maniere di passare parametri che il Quick Basic mette a disposizione. Infatti, premettendo ai nomi dei parametri la parola **BYVAL**, si passa il **valore** e non l'indirizzo di una determinata variabile e questo è necessario chiamando funzioni scritte in C; premettendo invece la parola **SEG**, i parametri vengono passati ancora per riferimento ma viene passato sia il segmento che l'offset (riferimento di tipo FAR). Se tutti i parametri devono essere passati in questo modo, è meglio usare l'istruzione CALLS.

*** CALLS**

Questa istruzione si comporta come la CALL ma passa tutti i parametri con l'indirizzo composto da segmento e offset; funziona come se con l'istruzione CALL si usasse il prefisso SEG in ogni argomento. La seguente frase:

```
CALLS ROUTINE(A, B, C)
```

è infatti equivalente alla seguente:

```
CALL ROUTINE(SEG A, SEG B, SEG C)
```

Utilizzando questa istruzione al posto della CALL nella chiamata delle Sub scritte in Basic, anche se non si dovesse notare alcun problema durante l'esecuzione di programmi nell'ambiente Quick Basic, si incorrerebbe in un sicuro errore compilando lo stesso programma con il compilatore BC.

*** CALL ABSOLUTE**

La routine ABSOLUTE, scritta in Assembler e fornita con Quick Basic nella libreria QB.LIB, è fatta per poter eseguire programmi in linguaggio macchina direttamente. Sebbene di uso molto più complesso delle istruzioni CALL e CALLS, che eseguono lo stesso lavoro più semplicemente, è necessaria per mantenere una certa compatibilità con programmi scritti in precedenza.

La sintassi prevede la chiamata della routine con la seguente linea:

CALL ABSOLUTE (par1, par2, …, locstart)

in cui par1, par2, … sono i parametri passabili alla routine e locstart è l'indirizzo della routine stessa. I parametri, se presenti, sono passati per riferimento near e gli offset relativi sono conservati nello stack. Ecco un esempio di uso di tale routine che realizza, tramite una chiamata al DOS, il calcolo dei bytes utilizzabili nel disco di lavoro:

```
'(Main Module)
DEFINT A-Z
CLS
DIM PROG(1 TO 14)
AOFF = VARPTR(PROG(1))
ASEG = VARSEG(PROG(1))
DEF SEG = ASEG
```

```
RESTORE ROUTINE
FOR T= 1 TO 28
        READ K
        POKE AOFF+T-1, K
NEXT T
CALL ABSOLUTE (BXS, CN, SXC, AOFF)
DEF SEG
PRINT "DRIVE DI DEFAULT… BYTES LIBERI = ";
PRINT CLNG(BXS) * CN * SXC
END

ROUTINE:
DATA &H55               ' PUSH BP
DATA &H89,&HE5          ' MOV SP,BP
DATA &HB4,&H36          ' MOV AH,36
DATA &HB2,&H00          ' MOV BL,0
DATA &HCD,&H21          ' INT 21
DATA &H8B,&H76,&H06     ' MOV SI,[BP+06]
DATA &H89,&H0C          ' MOV [SI],CX
DATA &H8B,&H76,&H08     ' MOV SI,[BP+06]
DATA &H89,&H1C          ' MOV [SI],BX
DATA &H8B,&H76,&H0A     ' MOV SI,[BP+0A]
DATA &H89,&H04          ' MOV [SI],AX
DATA &H5D               ' POP BP
DATA &HCA,&H06,&H00     ' RETF 06
```

Il programma, dopo aver dimensionato un array fatto da 14 elementi interi (28 bytes totali), memorizza in quest'ultimo una serie di valori che non sono altro che il programma in linguaggio macchina che, tramite la funzione 21h del DOS, sottofunzione 36h, ricava alcuni dati del disco di lavoro, e più precisamente:

- nel registro CX i bytes per ogni settore del disco
- nel registro AX i settori per ogni cluster del disco
- nel registro BX il numero di cluster liberi

Il valore dell'area libera si ottiene quindi moltiplicando i suddetti numeri tra loro, non prima di averli copiati all'interno degli indirizzi relativi alle tre variabili passate come parametri. Solo a questo punto, dopo un semplice calcolo, è possibile stampare il numero cercato. Si devono notare i seguenti fatti:

- i parametri sono passati per near reference; non è quindi possibile usare l'istruzione CALLS;

- l'indirizzo dell'ultimo parametro viene passato nello stack alla posizione 6, il secondo alla posizione 8 ed il primo alla posizione 0A; queste posizioni sono valide a patto che venga prima salvato nello stack il valore di SP e che questo venga copiato in BP; è rispetto a quest'ultimo registro che vengono considerate le posizioni dei parametri;

- all'uscita della routine, dopo aver ripreso il registro BP dallo stack, è necessario un ritorno di tipo FAR al programma chiamante, dato che la routine era stata chiamata con offset e segmento, avendo cura di predisporre un valore che permetta di riprendere il giusto indirizzo di rientro dallo stack; tale valore deve essere fatto seguire all'istruzione RETF ed è uguale al doppio del numero di argomenti passati dal programma. L'offset con cui il programma chiama la routine corrisponde al valore di locstart nella linea di comando (AOFF nell'esempio, che corrispondeva all'offset del primo elemento dell'array in cui è contenuto il programma in linguaggio macchina) ed il segmento è predisposto dall'istruzione DEF SEG che assegna il valore del segmento in cui è contenuto l'array;

- l'esecuzione del programma in ambiente Quick Basic è possibile solo se si entra nell'ambiente caricando la libreria Quick QB.QLB con il comando QB /LQB; con il compilatore BC è possibile eseguire il programma solo se al momento di usare il LINK si fornisce allo stesso il nome della libreria QB.LIB (comando completo : LINK PROABS,,NUL,QB); è infatti in queste librerie che la routine ABSOLUTE è definita.

*** CALL INT86OLD**

Vedere CALL INTERRUPT

*** CALL INT86XOLD**

Vedere CALL INTERRUPT

*** CALL INTERRUPT**

Anche se il QB prevede la CALL ABSOLUTE, la CALL INT86OLD e la CALL INT86XOLD come metodi per eseguire routines in assembler e per richiamare interrupts di sistema, un metodo più razionale e semplice di quest'ultimi è costituito dalla CALL INTERRUPT. Questa, come le suddette, è una routine fornita nella libreria QB.LIB ed ha la seguente sintassi:

CALL INTERRUPT (numinterrupt, regprima, regdopo)

in cui numinterrupt è un valore compreso tra 0 e 255 e rappresenta il numero della routine di interrupt che è possibile usare con la CPU 8088; queste routines sono predisposte dalla ROM del BIOS e dal DOS stesso ad eseguire determinati servizi generali. Le variabili che seguono (regprima e regdopo), sono di tipo REGTYPE definito all'interno del file include QB.BI anch'esso fornito con il Quick Basic. Tramite queste due variabili è possibile definire il contenuto di tutti i registri prima dell'esecuzione dell'interrupt e controllare il loro contenuto dopo l'esecuzione dello stesso; è possibile usare la stessa variabile per regprima e regdopo dato che i valori in entrata sono semplicemente sostituiti da quelli in uscita dall'interrupt ed i primi non sono più necessari. Il prossimo esempio illustra il modo in cui va usata la CALL INTERRUPT; per fare ciò si può risolvere il problema già affrontato con la CALL ABSOLUTE per vederne le differenze:

```
' $INCLUDE: 'QB.BI'
DIM REG AS REGTYPE
REG.AX=&H3600
REG.DX=0
CALL INTERRUPT (&H21, REG, REG)
L& = CLNG(REG.AX) * REG.BX * REG.CX
PRINT "DRIVE DI DEFAULT… BYTES LIBERI = ";L&
```

```
END
```

Con il metacomando $INCLUDE Quick Basic permette di utilizzare il tipo di dato definito al suo interno, tipo che è fondamentale per l'esecuzione di CALL INTERRUPT; nel tipo REGTYPE sono infatti presenti le definizioni dei registri della CPU, e questi è strutturato così:

```
TYPE REGTYPE
        AX      AS INTEGER
        BX      AS INTEGER
        CX      AS INTEGER
        DX      AS INTEGER
        BP      AS INTEGER
        SI      AS INTEGER
        DI      AS INTEGER
        FLAGS   AS INTEGER
END TYPE
```

In questo file è inoltre presente la dichiarazione (frase DECLARE) della routine INTERRUPT con i relativi parametri usati da quest'ultima.
Fatto questo si prepara una variabile, chiamata REG nell'esempio, di tipo REGTYPE e si preparano i valori che dovranno essere copiati nei registri della CPU prima dell'esecuzione dell'interrupt. Alla chiamata della routine INTERRUPT, l'interrupt 21 (esadecimale) subfunzione 36 (esadecimale) viene eseguito e questi ritorna nei registri della CPU dei valori che vengono ricopiati nella variabile REG. Questa funzione ritorna 3 numeri nel registro AX, BX e CX che moltiplicati fra loro, forniscono il numero dei bytes liberi sul disco di lavoro. Per eseguire questa operazione sugli altri dischi è sufficiente modificare il numero 0 assegnato al registro DX prima dell'esecuzione dell'interrupt, secondo questa tabella:

Valore in DX	Drive
0	Default
1	A
2	B
3	C
…	…

Per eseguire in ambiente Quick Basic o compilare il testo con il programma BC sotto DOS, attenersi alle regole specificate per la CALL ABSOLUTE.

*** CALL INTERRUPTX**

La CALL INTERRUPTX, a differenza della precedente, permette anche di usare i registri di segmento DS ed ES della CPU, consentendo quindi l'uso degli interrupt che li prevedono. Per questo, sempre nel file include QB.BI, sono presenti la definizione del tipo REGTYPEX e la frase di dichiarazione delle CALL INTERRUPTX, adatti all'utilizzo dei registri suddetti. Per il resto l'utilizzo di questa routine è identico a quello della CALL INTERRUPT, ed il seguente programma mostra un suo esempio d'uso:

```
' $INCLUDE: 'QB.BI'
```

```
DIM REG AS REGTYPEX
DIR$ = SPACE$(64)
REG.AX=&H4700
REG.DX=0
REG.SI=SADD(DIR$)
REG.DS=VARSEG(DIR$)
CALL INTERRUPTX (&H21, REG, REG)
ER = REG.FLAGS AND 1
IF ER THEN
        PRINT "ERRORE NUM. "; REG.AX; " …"
ELSE
        DIR$ = "\" + DIR$
        PRINT "DIREC. CORRENTE: "
END IF
END
```

Questo programma è utilizzato per ricavare, tramite la subfunzione 47 (esadecimale) dell'interrupt 21 (esadecimale) del DOS, la directory corrente del drive di lavoro. Cambiando il valore nel registro DX come nella tabella dell'esempio precedente, si può lavorare con gli altri drive. In uscita, controllando il primo bit del registro dei Flags (Carry bit), si può individuare un errore che il DOS ha incontrato nell'esecuzione dell'interrupt, altrimenti si ricava nella variabile DIR$, il cui indirizzo completo di segmento era stato passato all'inizio in DS:SI, il percorso attuale completo; il segno di radice iniziale non viene fornito da tale routine ed è per questo motivo che viene aggiunto prima di stampare il risultato. Un errore può esistere se viene fornito un numero di drive non corretto all'inizio.

Per eseguire in ambiente Quick Basic o compilare il testo con il programma BC sotto DOS, attenersi alle regole specificate per la CALL ABSOLUTE.

*** CASE**

Vedere SELECT CASE

*** CDECL**

Vedere DECLARE

*** CHAIN**

L'istruzione CHAIN è usata per concatenare programmi che fanno parte di un'unica procedura. È infatti molto conveniente dividere in diversi file i programmi complessi per eseguirne, a seconda delle necessità, uno di questi, è sufficiente fornire il nome dopo l'istruzione suddetta. Infatti, la sintassi di CHAIN è:

CHAIN nomefile

In cui nomefile è il nome del file che viene eseguito in catena. Con questa istruzione non viene chiuso alcun file eventualmente aperto e quindi questi possono essere condivisi. Il Quick Basic prevede che l'estensione del file sia .BAS se essi vengono eseguiti dentro l'ambiente, .EXE dopo che gli stessi vengono convertiti in file eseguibili. A differenza degli interpreti Basic che prevedevano opzioni come ALL per il passaggio delle variabili o con cui si poteva specificare il

numero di linea dal quale eseguire il programma concatenato, questo non è consentito in Quick Basic. Per quanto riguarda il passaggio delle variabili, esiste l'istruzione COMMON (a cui si rimanda per ulteriori chiarimenti), mentre al secondo problema si può ovviare usando delle variabili che fungano da 'semaforo'; ecco un esempio:

```
' (File P1.BAS)
COMMON K%(), PNT%
DEFINT A-Z
IF PNT=0 THEN
        CLS
        DIM K(100)
        PRINT "ELABORAZIONE DATI …"
        FOR T=1 TO 100
                K(T)=T*2
        NEXT T
        PRINT
        CHAIN "P2"
END IF
PRINT
PRINT "FINE ELABORAZIONE."
END

' (File P2.BAS)
COMMON K%(), PNT%
DEFINT A-Z
PRINT "STAMPA DATI …"
FOR T=1 TO 100
        PRINT K(T)
NEXT T
PRINT
PNT =1
CHAIN "P1"
```

In questo esempio, costituito da 2 file (P1.BAS e P2.BAS) che interagiscono per il riempimento e la stampa del contenuto di un vettore, la variabile PNT viene usata come flag che indica al file P1 l'avvenuta esecuzione del P2, e permette al primo di sincronizzarsi eseguendo le giuste istruzioni; infatti, in un primo momento, quando la variabile PNT è ancora a 0, viene eseguita la parte contenuta nel blocco IF…END IF di P1, mentre, una volta che la stessa variabile è a 1, e quindi è stato eseguito P2, viene eseguita la porzione all'esterno dello stesso blocco IF…END IF in P1.
Da notare che nessuna estensione viene specificata per P1 e P2 nelle istruzioni CHAIN dato che questa viene determinata da Quick Basic a seconda dell'ambiente in cui il programma viene eseguito. Inoltre, porre attenzione al fatto che, se compilati, i programmi devono usare la libreria BRUN40.LIB e non la BCOM40.LIB con la quale l'istruzione COMMON non funziona perché non supportata.

*** CHDIR**

In modo similare al comando del DOS, questa istruzione serve a cambiare la directory corrente di un determinato disco, ed ha la seguente sintassi:

CHDIR path

in cui path è una stringa, lunga al massimo 64 caratteri, che contiene il percorso completo necessario per cambiare directory di lavoro. È possibile includere il drive su cui agire prima del percorso separato dal simbolo ':'; in caso contrario l'istruzione agisce sul drive di lavoro. Attenzione ai seguenti fatti:

- questa istruzione non cambia il drive di lavoro ma solamente la directory di lavoro di un qualunque drive; il drive di lavoro rimane pertanto quello usato all'apertura di Quick Basic;

- non è possibile, con questa istruzione, conoscere la directory di lavoro di un disco, come nel caso in cui in DOS non si specifica alcun parametro dopo il comando; per ottenere ciò, riferirsi all'esempio dell'istruzione CALL INTERRUPTX;

- se, per un qualsiasi motivo, non fosse possibile cambiare directory nel drive specificato, viene emesso un messaggio di errore.

Nel prossimo esempio, si mostra come usare CHDIR usando delle variabili stringa:

```
DRIVE$ = "A:"
PATH$ = "\QB40\SORGENTI"
CHDIR DRIVE$ + PATH$
```

*** CIRCLE**

È l'istruzione usata in grafica per visualizzare circonferenze, ellissi e archi di cerchio. La sua sintassi è la seguente:

CIRCLE [STEP] (xc,yc),rag[,[col][,[angin][,[angfi][,aspetto]]]]

Dopo la parola chiave, il termine STEP può essere usato opzionalmente, per fare sì che le coordinate del centro siano intese relative alla posizione del cursore grafico determinata dall'ultima istruzione di tipo grafico usata; se tale termine non viene usato, le coordinate sono considerate assolute.
Seguono, tra parentesi tonde obbligatorie, le coordinate x e y, separate da una virgola, del centro della figura e poi il valore del raggio maggiore di quest'ultima; questi parametri sono obbligatori. Esiste poi una serie di parametri facoltativi che vengono usati in particolari situazioni; essi sono:

col è il colore con cui viene tracciata la figura. Per la scelta del valore di tale parametro è necessario conoscere le capacità della scheda video installata ed è utile riferirsi all'istruzione COLOR e all'istruzione SCREEN;

angin è il valore dell'angolo riferito alla circonferenza trigonometrica, dalla cui posizione parte il disegno della figura. È espresso in radianti da 0 a 2 pi-greco. Può essere negativo, fino a -2 pi-greco, ed in tale caso, viene tracciato un raggio fino alla posizione

dell'angolo inteso positivo; il suo valore di default è 0. Questo parametro, come il seguente, è necessario per tracciare degli archi;

angfi è il valore dell'angolo riferito alla circonferenza trigonometrica, alla cui posizione termina il disegno della figura. I valori che esso può assumere rispettano le regole imposte al parametro precedente. Il suo valore di default è 2 volte pi-greco;

aspetto è un valore adimensionale determinato dal rapporto tra coordinata y e x. Determina l'aspetto della ellissi visualizzata che, al limite, può diventare una circonferenza; per ottenere ciò, il valore si deve avvicinare al risultato della espressione seguente: (dimmony/divmonx) * (maxy/maxx) in cui maxx e maxy sono i valori massimi delle rispettive coordinate visualizzabili dalla scheda video installata e dimmonx e dimmony sono le dimensioni del video su cui vengono visualizzati i grafici; per molti monitors, questo rapporto è circa 4/3.

Per meglio comprendere il funzionamento di tale istruzione, ecco un esempio che comprende anche altre istruzioni di tipo grafico come LINE, PUT, GET:

```
SCREEN 1
P=3.141593
DEFINT A
DIM A1(450), A2(500), A3(500)
CX=30
CY=20
CIRCLE (CX,CY),20,1,-P*1.2,-P/1.3,.88
PAINT (CX+10,CY),2,1
LINE (CX-4,CY-10)-(CX-2,CY-8),,B
GET (CX-15,CY-20)-(CX+20,CY+20),A1
CLS
CIRCLE (CX,CY),20,1,-P*1.1,-P/1.1,.88
PAINT (CX+10,CY),2,1
LINE (CX-4,CY-10)-(CX-2,CY-8),,B
GET (CX-20,CY-20)-(CX+20,CY+20),A2
CLS
GET (CX-20,CY-20)-(CX+20,CY+20),A3
CLS
LINE (10,25)-(270,25),3
LINE (10,75)-(270,75),3
FOR T=270 TO 20 STEP -20
        CIRCLE (T,48),2,7
        PAINT (T,48),1,7
NEXT T
RITARDO=150
FOR T=270 TO 10 STEP -20
        PUT (T,30),A1,PSET
        FOR P=1 TO RITARDO
        NEXT P
        PUT (T,30),A3,PSET
        PUT (T,30),A2,PSET
```

```
            FOR P=1 TO RITARDO
            NEXT P
            PLAY "MBO4L64MSDEC"
            PUT (T,30),A3,PSET
    NEXT T
    END
```

Questo esempio è stato realizzato servendosi di una scheda grafica tipo CGA in modalità 320x200.

*** CLEAR**

Gli effetti di questa istruzione sono molteplici e più precisamente essa agisce nel seguente modo:

- svuota tutti i buffers di file eventualmente aperti e chiude questi ultimi;

- azzera tutte le variabili e gli arrays; pone tutte le variabili stringa uguali alla stringa nulla;

- opzionalmente, predispone un valore in bytes per lo stack del Quick Basic.

Proprio quest'ultima opzione è la più interessante e, per mantenere una certa compatibilità con le versioni del linguaggio Basic precedenti, è implementata con la seguente sintassi:

CLEAR [,,stack]

in cui le virgole separano argomenti fantasma, non usati da Quick Basic e stack è il valore suddetto. Tale valore deve essere cambiato, in genere aumentato, per programmi particolari che facciano molto uso di Sub, Function o che abbiano molti GOSUB..RETURN, FOR..NEXT, DO..WHILE, istruzioni queste che fanno largo uso dello stack. Ad esempio, il seguente programma che chiama ricorsivamente una routine per il calcolo del fattoriale, deve avere lo stack sovradimensionato, altrimenti viene visualizzato un errore di supero delle capacità dello stesso se si tenta di calcolare il fattoriale di un numero grande:

```
' (Main Module)
DECLARE FUNCTION FATTORIALE# (X%)
CLS
CLEAR ,,8000
FOR Z%=1 TO 170
    PRINT Z%; "! = "; FATTORIALE#(Z%)
NEXT Z%
END
' (Function FATTORIALE)
FUNCTION FATTORIALE# (X%)
    IF X%>1 THEN
            FATTORIALE# = X% * FATTORIALE# (X% - 1)
    ELSE
            FATTORIALE# = 1
    END IF
```

END FUNCTION

*** CLOSE**

È l'istruzione necessaria per chiudere un file in precedenza aperto con l'istruzione OPEN. La sua sintassi:

CLOSE [[#]numfile][,…]]

consente di chiudere in sequenza, con una istruzione, più file specificandone i numeri separati da virgole; senza alcun parametro, l'istruzione chiude tutti i file aperti al momento. Per i file in uscita, viene prima scaricato il contenuto del buffer. Il numero di file chiuso è nuovamente disponibile per essere usato da un altro file. Nel programma relativo all'istruzione suddetta, vengono mostrati diversi esempi di chiusura di file:

```
OPEN "O",#1,"PROVA1.DAT"
OPEN "O",#2,"PROVA2.DAT"
OPEN "O",#3,"PROVA3.DAT"
WRITE #1,"GOOD"
CLOSE #1
WRITE #2,"BYE"
WRITE #3,"QUICK BASIC"
CLOSE #2, #3
OPEN "I",#1,"PROVA1.DAT"
OPEN "I",#2,"PROVA2.DAT"
OPEN "I",#3,"PROVA3.DAT"
INPUT #1, S1$
INPUT #2, S2$
INPUT #3, S3$
PRINT S1$,S2$,S3$
CLOSE
```

Tenere presente che anche le istruzioni CLEAR, RESET, RUN e SYSTEM chiudono tutti i file aperti in precedenza.

*** CLS**

L'istruzione CLS è quella utilizzata per cancellare lo schermo ma ha subìto delle modifiche rispetto alle versioni precedenti del linguaggio, che ne hanno incrementato le potenzialità, e che, di contro, ne hanno reso l'uso un po' più complesso. La sua sintassi è infatti diventata la seguente:

CLS [{0 | 1 | 2}]

A seconda del valore che si pone dopo l'istruzione, o se questo manca, la cancellazione avviene su parti diverse dello schermo secondo delle regole che, a prima vista, possono confondere. Per riassumere i modi di funzionamento di CLS, sono utili le due tabelle seguenti:

se si sta lavorando in modo testo (SCREEN 0) e
non è stata eseguita alcuna VIEW PRINT

CLS Cancella tutto lo schermo
CLS 0 Cancella tutto lo schermo
CLS 1 Non agisce
CLS 2 Cancella tutto lo schermo tranne la riga 25

se si sta lavorando in modo testo (SCREEN 0) ed è stata eseguita una VIEW PRINT

CLS Cancella la finestra testo e la riga 25
CLS 0 Cancella tutto lo schermo
CLS 1 Non agisce
CLS 2 Cancella la finestra testo

se si sta lavorando in modo grafico (SCREEN 1..13) e non sono state eseguite VIEW né VIEW PRINT

CLS Cancella tutto lo schermo
CLS 0 Cancella tutto lo schermo
CLS 1 Cancella tutto lo schermo
CLS 2 Cancella tutto lo schermo tranne la riga 25

se si sta lavorando in modo grafico (SCREEN 1..13) ed è stata eseguita una VIEW PRINT

CLS Cancella tutto lo schermo
CLS 0 Cancella tutto lo schermo
CLS 1 Cancella tutto lo schermo
CLS 2 Cancella la finestra testo

se si sta lavorando in modo grafico (SCREEN 1..13) ed è stata eseguita una VIEW

CLS Cancella la finestra grafica
CLS 0 Cancella tutto lo schermo
CLS 1 Cancella la finestra grafica
CLS 2 Cancella tutto lo schermo tranne la riga 25

se si sta lavorando in modo grafico (SCREEN 1..13) e sono state eseguite una VIEW PRINT ed una VIEW

CLS Cancella la finestra grafica
CLS 0 Cancella tutto lo schermo
CLS 1 Cancella la finestra grafica

CLS 2   Cancella la finestra testo

Da tali tabelle si può notare che la CLS 0 cancella, comunque, tutto lo schermo, che la CLS 1 agisce solo in grafica e che, con la CLS 2 in modo grafico, si può cancellare anche una finestra di tipo testo; ecco un esempio di quest'ultimo uso:

```
SCREEN 1
LINE (10,10)-(100,100)
PRINT "PROVA CLS"
A$ = INPUT$(1)                  ' Attende un tasto
CLS 0                           ' Cancella tutto lo schermo
VIEW (100,100)-(150,150),2,1
LINE (1,1)-(50,50)
VIEW PRINT 2 TO 5
FOR T=2 TO 5
    LOCATE T,T
    PRINT "PROVA CLS"
NEXT T
A$ = INPUT$(1)                  ' Attende un tasto
CLS 1                           ' Cancella solo la finestra grafica
A$ = INPUT$(1)                  ' Attende un tasto
CLS 2                           ' Cancella solo la finestra testo
END
```

*** COLOR**

È l'istruzione usata per modificare i colori o gli attributi presenti sul video.
È una istruzione con una sintassi molto variabile, dipendente dal tipo di grafica attiva, dalla scheda video e dal monitor che si utilizzano. A seconda della modalità grafica impostata (istruzione SCREEN), la sua sintassi è una delle seguenti

(SCREEN 0)   **COLOR [primopiano][,[sfondo][,bordo]]**

(SCREEN 1)   **COLOR [sfondo][,palette]**

(SCREEN 7,8,9,10)   **COLOR [primopiano[,sfondo]]**

(SCREEN 11,12,13)   **COLOR [primopiano]**

Per la modalità 0, il colore di primopiano (del testo) può andare da 0 a 31 e più precisamente

0 = nero	1 = blu	2 = verde	3 = azzurro
4 = rosso	5 = magenta	6 = marrone	7 = bianco
8 = grigio	9 = azzurro	10 = verde ch.	11 = celeste
12 = rosso ch.	13 = magenta ch.	14 = giallo	15 = bianco lum.

per i valori oltre il 15, considerare gli stessi colori con, in più, il fatto d'essere lampeggianti: per esempio, il blu lampeggiante si ottiene con 1 + 16 = 17; per quanto riguarda invece il colore di sfondo, la scelta è limitata ai primi 8 colori che vanno dallo 0 al 7 e non è possibile usare il modo

lampeggiante; il bordo, infine, può essere di un colore compreso tra 0 e 15, ma non è supportato su schede tipo EGA, VGA o MGCA.

Per la modalità 1, usata se si dispone di una scheda CGA, lo sfondo è il colore dello schermo e può andare da 0 a 15; il secondo parametro è invece, un numero compreso tra 0 e 255, pari o dispari, che seleziona una tra due insiemi di colori (palette) composti, ognuno, da 4 colori:

Palette **0**

colore 0	colore di sfondo selezionato
colore 1	verde
colore 2	rosso
colore 3	marrone

Palette **1**

colore 0	colore di sfondo selezionato
colore 1	azzurro
colore 2	magenta
colore 3	bianco

Usando l'attributo 0 al quale corrisponde il colore di sfondo selezionato in precedenza, si può cancellare una figura disegnata in uno degli altri 3 colori dato che esso si confonderebbe con lo sfondo. In modalità 1 (con la scheda CGA), non possono comparire quindi più di 3 colori più quello di sfondo contemporaneamente sullo schermo ed il cambio di palette comporta un cambio immediato dei colori presenti sullo schermo secondo le tabelle suddette. Se si dispone di una scheda EGA o VGA, nella modalità schermo 1, è possibile assegnare a 4 attributi 16 colori diversi (con l'istruzione PALETTE).

Per la modalità 2, se supportata dalla scheda, si possono attribuire 16 colori a 2 attributi (0 e 1); non è supportata l'istruzione COLOR.

Per la modalità 3, l'istruzione COLOR non è eseguibile, dato che si tratta di una modalità monocromatica realizzata con la scheda Hercules.

La modalità 4, usata da alcuni modelli di computer Olivetti (M24, M240, M28, M280, M380) e dal modello 6300 della AT&T, può utilizzare 1 dei 16 colori disponibili per il testo ma il colore dello sfondo è comunque, il nero.

Nelle modalità 7 ed 8, con le schede EGA o VGA, si possono assegnare 16 colori a 16 attributi per poterli utilizzare contemporaneamente sullo schermo; le due modalità si differenziano per la risoluzione grafica.

Sempre in questo tipo di schede, se la capacità di memoria video delle stesse è di 64 K si possono utilizzare, nella modalità 9, fino a 16 colori assegnati a 4 attributi, mentre se la capacità è maggiore di 64 K, fino a 64 colori assegnati a 16 attributi.

La modalità 10, per le schede EGA o VGA, è usata con monitor monocromatici e permette l'utilizzo di 9 livelli di grigio assegnati a 4 attributi per simulare i colori.

Le modalità 11, 12 e 13 sono utilizzate con la scheda VGA e permettono di utilizzare fino a 262.144 colori assegnandoli, rispettivamente, a 2, 16 o 256 attributi. È quest'ultima la modalità più spettacolare, per quanto riguarda i colori, che il Quick Basic può gestire, con la scheda VGA.

Tutti gli assegnamenti dei colori agli attributi sono realizzati tramite l'istruzione PALETTE; le indicazioni sono comunque molto indicative, dato che esse possono differenziarsi a seconda della scheda e del monitor usato. In linea di massima, quindi, non esiste una regola per l'utilizzo dell'istruzione COLOR, ma essa va adeguata alle risorse hardware video disponibili.

Ecco un esempio di visualizzazione di rettangoli colorati in diversi colori, utilizzando la scheda CGA con la palette 0 e poi la palette 1

```
SCREEN 1
COLOR 0,0
LINE (50,50)-(80,80),1,BF           ' Colore verde
LINE (90,90)-(120,120),2,BF         ' Colore rosso
LINE (130,130)-(160,160),3,BF       ' Colore marrone
A$ = INPUT$(1)
COLOR ,1                            ' Cambia palette
                                    ' Verde = Azzurro
                                    ' Rosso = Magenta
                                    ' Marrone = Bianco
A$ = INPUT$(1)
```

*** COM**

Tramite questa istruzione è possibile intercettare degli eventi che si verificano in modo asincrono e che interessano una porta seriale di comunicazione. La sua sintassi prevede la specifica del numero di porta seriale da controllare e di un termine scelto tra ON, OFF e STOP:

COM(n) ON abilita l'intercettazione dei dati ricevuti su linea seriale, tramite una routine definita dall'istruzione ON COM(n) GOSUB. Dopo avere specificato il termine ON, se un carattere arriva alla porta seriale numero n, viene eseguita la routine specificata dopo il GOSUB.

COM(n) OFF disabilita la funzione di intercettazione dei dati ricevuti su linea seriale; la routine specificata dopo il GOSUB (nell'istruzione ON COM(n)..) non viene più eseguita e i dati eventualmente ricevuti, vengono persi.

COM(n) STOP agisce come la precedente opzione, ma i dati ricevuti, nei limiti del buffer di ricezione, sono conservati per potere eventualmente riprendere la loro elaborazione, tramite la COM(n) ON, in un secondo momento; questa opzione viene utilizzata per sospendere temporaneamente per brevi periodi, l'elaborazione dei dati ricevuti.

In generale, lo scheletro del programma tipo che usa tale istruzione per la porta seriale numero 1, è il seguente:

```
ON COM(1) GOSUB RECDATA
COM(1) ON                ' I dati sono processati
…
```

```
…
COM(1) STOP              ' I dati non sono processati ma conservati
…
…
COM(1) OFF               ' I dati non sono processati
END
RECDATA:
…                  ' Routine trattamento dati
RETURN
```

*** COMMON**

L'istruzione COMMON è, in realtà, una semplice dichiarazione di come alcune variabili debbano essere passate tra moduli diversi conservandone i valori. La sua sintassi è

COMMON [SHARED] [/blockname/] var [AS type] [, …]

Questa dichiarazione fa in modo che le variabili indicate nella lista siano memorizzate in un'area particolare di memoria (area COMMON), che consenta a tutti i moduli di accedere alle stesse. Il termine SHARED è necessario quando le variabili devono essere condivise, oltre che tra due moduli, anche con le Subs e le Functions di questi, evitando la dichiarazione SHARED in ogni Sub o Function. Molte volte, più semplicemente, la frase COMMON è usata appunto per rendere alcune variabili di tipo globale e cioè fare in modo che esse siano disponibili sia a livello di modulo che a qualsiasi livello di Sub o Function, come nell'esempio che segue:

```
COMMON SHARED GIORNO%, MESE%, ANNO%
DECLARE SUB S1(X!)
DECLARE SUB S2(Y!)
DECLARE FUNCTION F1!(U!)
GIORNO = 10
MESE = 10
ANNO = 2000
…
…
END
SUB S1(X)
…
END SUB
SUB S2(y)
…
END SUB
FUNCTION F1(U)
…
END FUNCTION
```

Nel caso precedente, le variabili GIORNO, MESE ed ANNO, tutte intere, sono condivise con la Sub S1, con la Sub S2 e con la Function F1, evitando che in questa venga posta l'istruzione SHARED; il fatto che le variabili condivise siano intere, viene dichiarato dal simbolo % relativo a questo tipo di dati; si sarebbe anche potuto usare la clausola AS INTEGER ma, come per tutti gli

altri tipi definiti in Basic, questo modo di operare genererebbe linee molto lunghe; è invece necessaria tale scrittura quando le variabili interessate sono di tipo definito dall'utente. Per quanto riguarda gli arrays, essi vengono passati con il loro nome e tipo e con le doppie parentesi tonde aperta-chiusa; per passare, ad esempio, l'array A intero dimensionato a 10 x 20 elementi, sarebbero necessarie le due frasi

```
DIM A%(1 TO 10, 1 TO 20)
COMMON SHARED A%( )
```

Quando, invece, si vogliono passare dei valori tra moduli diversi, facendo in modo che ogni modulo riceva le variabili che gli interessano, si usa il /blockname/. Questo è un identificatore che fa in modo che alcune variabili appartengano ad un blocco definito e che tale blocco, identificato da un nome, sia passato ad un altro modulo; per meglio comprendere tale modo di operare è consigliabile riferirsi al seguente programma di esempio:

```
' (Main Module)
DECLARE SUB CALCVOLUME ( )
DECLARE SUB CALCPESO ( )
COMMON /DATIVOLUME/ SPIGOLO, VOLUME
COMMON /DATIPESO/ DENSITA, PESO
CLS
INPUT "SPIGOLO DEL CUBO IN CM. ", SPIGOLO
INPUT "DENSITA' DEL CUBO IN GRAMMI/CM CUBO ", DENSITA
PRINT
CALCVOLUME
CALCPESO
PRINT "IL VOLUME E' "; VOLUME; " CM. CUBICI"
PRINT "IL PESO E' "; PESO; " GRAMMI"
END
' (Module CALCMOD.BAS)
COMMON SHARED /DATIVOLUME/ SPIGOLO, VOLUME
COMMON SHARED /DATIPESO/ DENSITA, PESO
' (Sub CALCPESO)
SUB CALCPESO
      PESO = DENSITA * VOLUME
END SUB
' (Sub CALCVOLUME)
SUB CALCVOLUME
      VOLUME = SPIGOLO ^ 3
END SUB
```

Notare che le due Subs richiamate nel primo modulo, appartengono al secondo e che le frasi COMMON necessarie per il collegamento, sono sempre presenti a livello di modulo; per potere creare il secondo modulo, riferirsi all'opzione Create File (Module) del Menu File.

Mentre i blocchi COMMON già visti sono chiamati 'con nome', i blocchi COMMON 'vuoti' devono essere utilizzati in certe situazioni; un blocco COMMON vuoto non utilizza il parametro 'blockname' e ciò si rende necessario quando si utilizza l'istruzione CHAIN. Infatti, per potere

passare il contenuto di variabili o arrays tra file diversi concatenati tramite l'istruzione CHAIN, non è possibile usare i blocchi COMMON con nome, come mostra il seguente esempio

```
‘ (Main Module S1.BAS)
COMMON GIORNO%, MESE%, ANNO%, OPZ%
CLS
GIORNO%=1
MESE%=1
ANNO%=2000
INPUT “OPZIONE “, OPZ%
CHAIN “S2”
‘ (Main Module S2.BAS)
COMMON G%, M%, A%, O%
CLS
PRINT G%, M%, A%
PRINT “OPZIONE “; O%
END
```

Come nell'esempio precedente, si può notare che non sono importanti i nomi dati alle variabili nei due moduli per permettere il passaggio dei valori, ma la loro posizione ed il loro tipo; il passaggio, infatti, avviene tramite la lettura di quanto era stato memorizzato nell'area COMMON dal programma chiamante, da parte di quello chiamato, nella stessa identica sequenza di come i dati sono stati scritti; ecco dunque che se i dati non sono dello stesso tipo o sono in posizione diversa, essi non possono avere lo stesso valore di quelli di partenza; per quanto riguarda il numero dei dati nel programma chiamato, essi possono anche essere in più o in meno; nel primo caso, le variabili in eccesso sono poste a zero (le numeriche) o vuote (le stringhe), nel secondo, le variabili non considerate vengono ignorate.
Quando si realizza una procedura una procedura tramite molti file concatenati con l'istruzione CHAIN, è buona norma evitare che un eventuale richiamo di un file che non sia il principale, possa far partire la stessa da un punto non corretto; nell'ultimo programma, ad esempio, dopo averlo compilato, richiamando il file S2.EXE, sarebbe stato possibile eseguirlo arrivando a risultati sbagliati; per rimediare, tramite la dichiarazione COMMON, predisporre una variabile di controllo in ogni file concatenato, nel seguente modo

```
‘ (Main Module S1.BAS)
COMMON GIORNO%, MESE%, ANNO%, OPZ%, PSW&
CLS
GIORNO%=1
MESE%=1
ANNO%=2000
INPUT “OPZIONE “, OPZ%
PSW&=999999
CHAIN “S2”
‘ (Main Module S2.BAS)
COMMON G%, M%, A%, O%, PSW&
IF PSW& <> 999999 THEN
        CHAIN “S1”
END IF
```

```
CLS
PRINT G%, M%, A%
PRINT "OPZIONE "; O%
END
```

Per ultimo, considerare il fatto che l'istruzione COMMON non è supportata dalla libreria BCOM40.LIB ma solo dalla BRUN40.LIB; non è possibile compilare programmi che la usano con l'opzione /O e linkarli con la libreria BCOM40.LIB per renderli indipendenti dal BRUN40.EXE; è necessario quindi, compilarli in maniera tradizionale e linkarli con la libreria BRUN40.LIB (di default) per poi eseguirli con il run-time module BRUN40.EXE-

*** CONST**

Con la dichiarazione CONST si possono specificare una serie di costanti usate nel programma. La definizione di costanti ed il loro uso, è un metodo migliore rispetto all'utilizzo di variabili che occupano memoria e fanno sprecare più tempo nell'esecuzione del programma stesso. Quick Basic permette di definire delle costanti il cui tipo può essere esplicitamente dichiarato dopo il nome delle stesse tramite il consueto simbolo che distingue i numeri nelle loro precisioni dalle stringhe (%, !, #, $). Nel caso in cui manchi tale simbolo, il tipo delle costanti viene definito dal tipo del risultato dell'espressione che segue e non dalle frasi di tipo DEFtipo che valgono solo per le variabili e gli arrays. Comunque, il nome delle costanti è considerato a prescindere dal loro tipo e quindi non va specificato il simbolo corrispondente durante il loro uso. Non possono esistere altre variabili con il nome di costanti e, cosa che rende ancora più utili le costanti, se dichiarate al livello del modulo, esse sono definite in qualunque parte dello stesso comprese le sue Subs e Functions; se dichiarate in quest'ultime, sono di tipo locale. Ecco un esempio d'uso delle costanti:

```
CONST PIGRECO=3.1415926565#
CONST VLIM = .8, COND = "SENO DI R > 0.8"
FOR R=0 TO PIGRECO
        K=SIN(R)
        PRINT T, K
        IF K>VLIM THEN
                PRINT COND
        ELSE
                PRINT
        END IF
NEXT K
END
```

*** DATA**

Viene usata per dichiarare un insieme di dati costanti che non sono ancora assegnati ad alcun identificatore; essi possono essere letti e memorizzati all'interno di variabili semplici o arrays (tal dati si possono considerare come appartenenti ad un file sequenziale auto contenuto nel programma).

La sintassi prevede la separazione delle costanti tra loro tramite la virgola; se quest'ultima deve apparire come parte di una costante alfanumerica, la stessa deve essere delimitata dalle virgolette; se la costante alfanumerica deve contenere spazi iniziali, spazi finali ed altri caratteri speciali, si agisce come nel caso precedente. Se la virgola di separazione viene seguita

immediatamente da un’altra virgola, il dato definito viene interpretato come zero (**se numerico**) o come stringa vuota (**se alfanumerico**). Non è possibile riferirsi al contenuto di variabili o costanti definite in precedenza, scrivendo il loro nome nelle frasi DATA.
Le frasi DATA sono considerate in sequenza e l’ordine con cui l’istruzione READ li legge è il seguente:

- all’inizio, la prima istruzione READ legge la prima frase DATA;

- in seguito, tramite l’istruzione RESTORE, è possibile che la READ rilegga frasi DATA già lette.

Il seguente esempio mostra come è possibile scrivere la data corrente tramite l’uso delle frasi DATA e l’uso della CALL INTERRUPT:

```
‘ $INCLUDE: ‘QB.BI’
DEFINT A-Z
CLS
DIM REG AS REGTYPE
DIM MESE$(1 TO 12), GSETT$(0 TO 6)
FOR T=1 TO 12
  READ MESE$(T)
NEXT T
FOR T=0 TO 6
  READ GSETT$(T)
NEXT T
REG.AX=&H2A00
CALL INTERRUPT(&H21, REG, REG)
GSETT=REG.AX AND &HFF
GIORNO=REG.DX AND &HFF
MESE=REG.DX \ &H100
ANNO=REG.CX
PRINT “Oggi e’ “;
PRINT GSETT$(GSETT); GIORNO; MESE$(MESE); “ “; ANNO
END

DATA Gennaio,Febbraio,Marzo,Aprile,Maggio,Giugno,Luglio
DATA Agosto,Settembre,Ottobre,Novembre,Dicembre
DATA Domenica,Lunedi’,Martedi’,Mercoledi’
DATA Giovedi’,Venerdi’,Sabato
```

Per eseguire in ambiente Quick Basic o compilare il testo con il programma BC sotto DOS, attenersi alle regole specificate per la CALL ABSOLUTE.

*** DATE$**

Questa istruzione, che fa da complemento alla funzione omonima, è utilizzata per predisporre la data di sistema come fa il comando DATE del DOS. È da considerarsi piuttosto come una variabile riservata del Quick Basic e la sintassi per il suo uso è la seguente:

DATE$ = data

in cui 'data' è una stringa in cui viene specificata la nuova data di sistema, in uno dei seguenti formati:

mm/gg/aa   mm/gg/aaaa mm-gg-aa   mm-gg-aaaa

Un uso errato di questa istruzione provoca l'errore Illegal Function Call di Quick Basic; attenzione al fatto che la data di sistema, per il DOS, è valida nei limiti 01/01/1980 e 31/12/2099. Se, per esempio, durante l'esecuzione di un programma, si volesse immettere la data di sistema, il programma tipo sarebbe:

```
     ON ERROR GOTO ERTRAP
     CLS
     …
100  INPUT "Data di sistema (gg/mm/aa) ", D$
     D$ = MID$(D$, 4, 3) + LEFT$(D$, 3) + MID$(D$, 7)
101  DATE$ = D$
     …
     END
     ERTRAP:
        IF ERR=5 AND ERL=101 THEN
               PRINT "Data illegale."
        RESUME 100
        END IF
        ON ERROR GOTO 0
```

*** DECLARE**

La frase di dichiarazione DECLARE è necessaria in Quick Basic quando si utilizzano delle Subs o delle Functions definite dall'utente. Le Subs e le Functions sono memorizzate nel file dopo il modulo che, generalmente, le utilizza e quindi, durante la compilazione, si avrebbero degli errori dato che esse verrebbero chiamate prima della loro definizione. È per questo che, all'inizio del modulo, le frasi DECLARE vengono poste in maniera da descrivere il nome, il tipo e gli argomenti usati dalla Function o dalla Sub; contemporaneamente viene abilitato il controllo del tipo e del numero degli argomenti passati ad ogni Sub o Function, ad ogni chiamata delle stesse. La frase DECLARE viene generata automaticamente al momento della registrazione su file per ogni Sub o Function per le quali manchi; esistono però dei casi in cui la frase non è generata automaticamente , e cioè:

- quando una Function non appartiene al modulo che la chiama e il modulo in cui essa è registrata non è caricato in memoria;

- quando una Sub è definita in un altro modulo che non sia quello in cui viene chiamata, anche se questo è caricato in memoria; in questo caso, però, la frase DECLARE non è necessaria, a meno che la Sub non venga richiamata con l'istruzione CALL;

- quando una Sub o una Function è definita in una libreria.

In questi casi è necessario che la frase DECLARE venga inserita appositamente.

Per quanto riguarda la sintassi della dichiarazione DECLARE, essa si differenzia a seconda che le procedure richiamate siano scritte in Quick Basic o in un altro linguaggio; nel primo caso, la sintassi è

DECLARE {**SUB** | **FUNCTION**} **nomeproc** [**(**[**elenco parametri**]**)**]

in cui, dopo la parola DECLARE, si deve scegliere la parola FUNCTION o la parola SUB a seconda che la procedura ritorni un valore o no; segue il nome della procedura che può essere lungo fino a 40 caratteri alfabetici e numerici e che non deve contenere spazi ed altri caratteri speciali; solo per la Function, il nome della stessa può terminare con uno dei simboli che indichi il tipo di valore restituito (%, &, !, #, $); per ultimo è posto un elenco dei parametri, eventualmente presenti, racchiusi tra parentesi tonde, che deve essere passato alla procedura specificata; in questo elenco, devono essere specificate le variabili con il tipo separate da una virgola in uno dei seguenti modi

DECLARE SUB PROVA (A%, B!, C$)
oppure
DECLARE SUB PROVA (A AS INTEGER, B AS SINGLE, C AS STRING)

se si volessero passare, ad esempio, 3 parametri dei tipi specificati; è infatti possibile usare, sia i simboli specifici dei tipi, o la forma AS type in cui ‘type’ può essere uno tra i tipi standard (INTEGER, LONG, SINGLE, DOUBLE o STRING), o un tipo definito dall’utente con l’istruzione TYPE..END TYPE per cui quest’ultima forma è obbligatoria.
Se non si dovessero passare parametri, si dovrebbero solo specificare le parentesi vuote come ad esempio

DECLARE SUB PROVA ( )

mentre, per passare un array, al suo nome si deve fare seguire la coppia di parentesi tonde vuote

DECLARE SUB PROVA (A%( ))
oppure
DECLARE SUB PROVA (A( ) AS INTEGER)

Per quanto riguarda le stringhe a lunghezza fissa, bisogna fare attenzione al fatto che esse non possono essere passate come argomento e quindi, esse sono automaticamente convertite in stringhe a lunghezza variabile prima del passaggio stesso.

Per quanto riguarda il passaggio di parametri a procedure scritte in altri linguaggi, la sintassi prevede la seguente forma

DECLARE {**SUB** | **FUNCTION**} **nomep** [**CDECL**] [**ALIAS “alias”**] [**(**[**elpar**]**)**]

in cui compaiono due elementi in più, e più precisamente

- CDECL, che è usata per precisare che il passaggio dei parametri avviene secondo le convenzioni del linguaggio C e cioè partendo dal parametro a destra verso sinistra; è necessario specificare questa opzione quando deve essere richiamata una procedura scritta in questo linguaggio, anche se questo accorgimento non è l'unico per interfacciare il linguaggio C al Quick Basic;

- ALIAS "alias", che deve essere specificato mettendo tra virgolette il nome della procedura scritta in un altro linguaggio, quando questo nome è diverso da quello indicato dopo la dichiarazione DECLARE; nel caso del linguaggio C, nel file oggetto generato da un sorgente scritto in tale linguaggio, le funzioni sono precedute dal simbolo underscore (_) ma questo viene aggiunto automaticamente quando si aggiunge la parola CDECL; se il nome fosse diverso in un'altra parte dello stesso, bisognerebbe specificarlo includendo questo carattere; ad esempio, se si dovesse dichiarare una funzione PROVA senza parametri che, nel sorgente C, ha nome **_progprova**, si dovrebbe specificare la seguente frase DECLARE

DECLARE FUNCTION PROVA CDECL ALIAS "_progprova" ( )

Inoltre, nell'elenco dei parametri che, normalmente, vengono passati per *near reference* (con il solito offset), per il linguaggio C è necessario specificare la clausola BYVAL prima di ogni argomento, per permettere di passare gli stessi per valore e non per indirizzo, come appunto richiede il linguaggio. Quick Basic permette di utilizzare anche altre clausole che modificano la maniera di passare i parametri, tanto che la sintassi dell'elenco dei parametri, assume la seguente forma

([{**BYVAL | SEG**}] **var** [**AS type**][**, …**]**)**

in cui la clausola SEG permette di passare gli argomenti per *far reference* (con il segmento e l'offset) per i linguaggi che lo richiedano; in mancanza di una di queste due clausole, il parametro è passato per near reference, come già specificato.

Bisogna tener presente, nel voler utilizzare dei programmi scritti in C con Quick Basic, le seguenti regole:

- in C possono esistere solo delle funzioni; una Sub può essere quindi dichiarata in Quick Basic solo se la relativa funzione in C è di tipo void (che non ritorna valore);

- il programma in C non deve comprendere il main, ma solo le funzioni richieste e deve essere compilato usando il modello medio di indirizzamento della memoria (opzione /AM oppure /Mm con il compilatore MSC 6.0 Microsoft);

- bisogna usare l'opzione /c perché il compilatore generi il file oggetto senza tentare di linkarlo (non si deve generare un file eseguibile dal sorgente scritto in C); inoltre, è molte volte necessario usare l'opzione /Gs per evitare che la procedura scritta in C operi dei controlli sullo stack che interferiscono con QB;

- per linkare il file oggetto generato da Quick Basic con quello generato da C, si deve usare l'opzione /NOE del link per permettere l'uso delle definizioni multiple di simboli che, inevitabilmente, occorrono in queste circostanze;

- è infine necessaria, per il link, la libreria MLIBCE.LIB di MSC 6.0, quando si utilizzino funzioni C predefinite.

Ad esempio, ecco un programma di prova per l'interfacciamento con il linguaggio C che calcola, tramite la funzione hypot di MSC 6.0, l'ipotenusa di una serie di triangoli rettangoli di cui siano conosciuti i cateti; il programma in Quick Basic è

```
DECLARE FUNCTION IPOTENUSA# CDECL (BYVAL C1#, BYVAL C2#)
CLS
FOR CAT1#=1 TO 20
    FOR CAT2#=40 TO 20 STEP -1
            PRINT "CATETO 1 : "; CAT1#;
            PRINT "CATETO 2 : "; CAT2#;
            PRINT "IPOTENUSA : "; IPOTENUSA#(CAT1#, CAT2#)
    NEXT CAT2#
NEXT CAT1#
END
```

ed il programma scritto in C

```
#include <math.h>
double ipotenusa(x, y)
double x,y;
{
    Return(hypot(x,y));
}
```

Il programma in C, chiamato HYP.C, sarà compilato con il comando

CL /c /Mm /Gs hyp.c

che provvede a generare il file oggetto (HYP.OBJ) necessario, usando il modello medio di indirizzamento delle memoria, mentre il programma in Quick Basic, PROVA.BAS, con la linea

BC PROVA;

che genera il file PROVA.OBJ; per ultima, la fase del link, viene realizzata tramite il comando

LINK /NOE PROVA+HYP;

che genera il file PROVA.EXE eseguibile sotto DOS.
Per potere eseguire il programma PROVA.BAS in ambiente Quick Basic, è necessario generare una libreria di tipo Quick da caricare con il comando QB all'atto della partenza del Quick Basic; devono essere eseguite le seguenti operazioni:

1. generazione della libreria PRO.LIB dal file HYP.OBJ tramite il comando **LIB PRO +HYP;** dato sotto DOS;

2. generazione della Quick Library PRO.QLB dal file PRO.LIB tramite il comando **LINK /Q PRO.LIB,PRO.QLB,NUL,BQLB40;** sotto DOS;

3. avvio del programma PROVA.BAS in ambiente Quick Basic con il comando **QB /LPRO /RUNPROVA** che provvede a caricare la libreria contenente la funzione IPOTENUSA.

*** DEFDBL**

È l'istruzione che provvede a dichiarare di tipo DOUBLE un insieme di variabili specificate; la sua sintassi prevede che vengano specificate le iniziali delle variabili interessate separate da una virgola; se si separano le iniziali con un trattino, tutte quelle intermedie sono interessate; ad esempio, per dichiarare di tipo DOUBLE le variabili che cominciano per A, E, K, L ed M si può dare una delle seguenti istruzioni equivalenti

```
DEFDBL A,E,K,L,M
```

oppure

```
DEFDBL A,E,K-M
```

Una volta dichiarato il tipo DOUBLE, le variabili che iniziano con una delle lettere interessate, non necessitano del simbolo # per potere operare con la precisione richiesta; questo vale anche per gli arrays.

*** DEF FN**

Tramite tale dichiarazione si possono definire delle funzioni utente nel modo in cui il Basic effettua tale operazione anche in versioni più vecchie di Quick Basic; in effetti tale metodo presenta degli inconvenienti rispetto alle Functions e non è tanto flessibile quanto quest'ultime; la dichiarazione si può presentare in due forme; secondo la prima sintassi, quella tradizionale, essa va scritta nel modo seguente

DEF FNnomefunzione(elencoparametri) = espressione

in cui il nome della funzione può essere lungo fino a 40 caratteri e rispetta le regole di formazione dei nomi delle variabili; la funzione restituisce un valore del tipo specificato dopo il nome con uno dei simboli tradizionali (%, &, !, #, $); nell'elenco dei parametri, il tipo delle variabili può essere specificato con uno dei simboli suddetti o nella forma AS type in cui 'type' è una delle parole INTEGER, LONG, SINGLE, DOUBLE o STRING. Nell'espressione sono validi tutti gli operatori aritmetici, logici e di relazione standard del Quick Basic. Per realizzare una funzione che restituisca il massimo tra due valori in doppia precisione, ecco un programma che può servire da esempio

```
DEF FNMAX#(A#, B#) = -(A#>B#)*A#-(A#<=B#)*B#
CLS
INPUT “PRIMO NUMERO : “,L1#
INPUT “SECONDO NUMERO : “,L2#
PRINT “IL PIU’ GRANDE E’ “; FNMAX#(L1#, L2#)
END
```

Con la seconda sintassi della dichiarazione DEF FN, è possibile usare quasi tutte le istruzioni del linguaggio, dato che la funzione viene definita in un blocco che ha un inizio ed una fine; questi sono segnati dall'istruzione DEF FN e dall'istruzione END DEF; per realizzare, con questa sintassi, l'esempio precedente, sono sufficienti le seguenti linee

```
DEF FNMAX#(A#,B#)
        IF A#>B# THEN
                FNMAX#=A#
        ELSE
                FNMAX#=B#
        END IF
END DEF
CLS
INPUT "PRIMO NUMERO : ",L1#
INPUT "SECONDO NUMERO : ",L2#
PRINT "IL PIU' GRANDE E' "; FNMAX#(L1#, L2#)
END
```

Notare come, in questo caso, il valore che deve ritornare la funzione sia attribuito nel blocco al nome stesso della funzione (FNMAX#=A# o FNMAX#=B#), a seconda dei casi; se il calcolo della funzione deve terminare prima della END DEF, si può usare l'istruzione EXIT DEF che evita l'esecuzione del resto della funzione; ad esempio, nel programma

```
DEF FNR4#(X#)
        IF X#<0 THEN
                FNR4#=-1
                EXIT DEF
        END IF
        FNR4#=X#^(1/4)
END DEF
CLS
FOR T#=-5 TO 5
        RQ#=FNR4#(T#)
        PRINT "RAD. QUARTA DI "; T#; " = ";
        IF RQ#=-1 THEN
                PRINT "** Argomento errato **"
        ELSE
                PRINT RQ#
        END IF
NEXT T#
END
```

per tutti i valori di X minori di 0, la funzione che calcola la radice quarta di X, ritorna un valore -1 di segnalazione d'errore, mentre per tutti gli altri calcola il risultato.

Tenere presente che le funzioni definite con DEF FN non possono essere nidificate e non possono essere ricorsive; inoltre esse sono definite solo a livello di modulo e sono visibili solo nel modulo in cui sono definite; inoltre è possibile avere degli inconvenienti con l'uso di

istruzioni di I/O da file e con istruzioni grafiche se usate in una DEF FN..END DEF, a seconda di come il compilatore ottimizza il codice oggetto; sono questi i motivi per cui è preferibile utilizzare le Functions al posto delle DEF FN..END DEF nei programmi.

*** DEFINT**

È l'istruzione che provvede a dichiarare di tipo INTEGER un insieme di variabili specificate; la sintassi prevede che vengano specificate le iniziali delle variabili interessate separate da una virgola; se si separano le iniziali con un trattino, vengono incluse tutte quelle intermedie; ad esempio, per dichiarare di tipo INTEGER le variabili che cominciano per I, J, K, L si può dare la seguente istruzione

DEFINT I-L

Quanto detto per l'istruzione DEFDBL, vale anche per la DEFINT.

*** DEFLNG**

È l'istruzione che provvede a dichiarare di tipo LONG un insieme di variabili specificate; la sintassi prevede che vengano specificate le iniziali delle variabili interessate separate da una virgola; se si separano le iniziali con un trattino, tutte quelle comprese tra le lettere specificate, sono incluse nella dichiarazione; ad esempio, per dichiarare di tipo LONG le variabili che cominciano per A, B, C e K si può dare una delle seguenti istruzioni equivalenti

DEFLNG A,B,C,K
oppure
DEFLNG A-C,K

Quanto inoltre specificato per l'istruzione DEFDBL, vale anche per la DEFLNG.

*** DEF SEG**

Questa istruzione viene usata per definire un segmento di memoria che deve essere utilizzato da alcune istruzioni del Quick Basic come CALL ABSOLUTE, PEEK, POKE, BLOAD e BSAVE; normalmente il segmento è quello di default dei dati di Quick Basic (dove sono conservate le variabili), ma per motivi particolari, esso può essere cambiato rispettando la seguente sintassi dell'istruzione

DEF SEG [**=segmento**]

Se non viene specificato il segmento, utilizzando la sola istruzione DEF SEG, ritorna ad usare il segmento dei dati di default per le istruzioni suddette; un esempio di utilizzo di tale istruzione, è mostrato nell'esempio seguente, che legge il valore di una locazione (esadecimale 417) del segmento 0, in cui viene depositato continuamente un valore che dipende dal fatto che i tasti shift siano premuti o no:

```
DEFINT A-Z
DO
    DEF SEG=0
    P=PEEK(&H417) AND 3
    DEF SEG
```

```
SELECT CASE P
        CASE 1
                PRINT "Shift destro premuto"
        CASE 2
                PRINT "Shift sinistro premuto"
        CASE 3
                PRINT "I due Shift premuti"
        CASE ELSE
                PRINT "Nessuno Shift premuto"
END SELECT
LOOP
```

*** DEFSNG**

È l'istruzione che provvede a dichiarare di tipo SINGLE un insieme di variabili specificate; la sintassi prevede che vengano specificate le iniziali delle variabili interessate separate da una virgola; se si separano le iniziali con un trattino, sono incluse nella dichiarazione tutte quelle intermedie; ad esempio, per dichiarare di tipo SINGLE le variabili che cominciano per X, Y e Z si può dare una delle seguenti istruzioni equivalenti

DEFSNG X,Y,Z
oppure
DEFSNG X-Z

Notare che, senza alcuna istruzione di tipo DEFtype, per default, tutte le variabili sono intese di tipo SINGLE; per questo motivo, la dichiarazione DEFSNG è usata solo in determinate circostanze.
Quanto inoltre specificato per l'istruzione DEFDBL vale anche per DEFSNG.

*** DEFSTR**

È l'istruzione che provvede a dichiarare di tipo STRING un insieme di variabili specificate; la sintassi prevede che vengano specificate le iniziali delle variabili interessate separate da una virgola; se si separano le iniziali con un trattino, tutte quelle intermedie sono interessate; ad esempio, per dichiarare di tipo SINGLE le variabili che cominciano per S, T, U e Z si può dare una delle seguenti istruzioni equivalenti

DEFSTR S,T,U,Z
oppure
DEFSTR S-U,Z

Quanto detto per l'istruzione DEFDBL vale anche per DEFSTR.
Porre particolare attenzione nell'uso di questa istruzione, al fatto che, dopo averla specificata, è perfettamente legale scrivere frasi del tipo

T = "Questa e' una prova"

che non generano errori del tipo "Type mismatch"; le stringhe interessate sono sempre intese a lunghezza variabili.

*** DIM**

È la classica istruzione usata per preparare lo spazio ed inizializzare gli arrays; in Quick Basic è anche utilizzata per dichiarare le variabili e gli arrays di tipo utente; la sua sintassi è la seguente

DIM [SHARED] var [(([indin TO] indfi)] [AS type] [,…]

in cui il termine SHARED è usato quando si vuole che le variabili o gli arrays dimensionati siano condivisi da tutte le Subs e le Functions del modulo in cui si trova; 'var' è il nome della variabile semplice o dell'array che si vuole inizializzare, nome che può essere lungo fino a 40 caratteri e che non deve contenere spazi; se la variabile è semplice, non viene specificato alcun indice in seguito, altrimenti, nel caso degli arrays, è possibile indicare l'intervento degli indici, nei due seguenti modi

DIM A(100)
oppure
DIM A(0 TO 100)

Tutte e due le forme sono corrette ed equivalenti, ma, nel secondo caso, con la clausola TO, si specifica l'indice minimo ed il massimo; senza tale clausola, è solo specificato l'indica massimo e quello minimo è 0, a meno che non sia stata eseguita l'istruzione OPTION BASE 1, nel qual caso è 1. L'indice, in Quick Basic, può anche essere negativo; infatti i suoi limiti sono -32768..32767; nel caso di array a più dimensioni, gli indici minimi e massimi di queste ultime, vanno separati da una virgola, come nell'esempio seguente in cui si dimensiona una matrice tridimensionale di 10 x 20 x 15 elementi

DIM MAT(1 TO 10, 1 TO 20, 1 TO 15)

Tenere presente che il numero massimo di dimensioni che un array può avere, è 60.

Il tipo assegnato alla variabile, sia essa semplice o array, è definito dall'ultima istruzione DEFtype eseguita ed, in mancanza di quest'ultima, è per default SINGLE; esso può essere modificato tramite l'uso di uno dei simboli caratteristici di Quick Basic dopo il nome della variabile o array; in alternativa, può essere specificata la clausola AS seguita dai termini INTEGER, LONG, SINGLE, DOUBLE o STRING a seconda dei casi, come ad esempio

DIM MAT#(1 TO 100, 1 TO 50)
oppure
DIM MAT(1 TO 100, 1 TO 50) AS DOUBLE

Per quanto riguarda le stringhe a lunghezza fissa, esse sono definite con l'istruzione DIM specificando la loro lunghezza dopo l'attributo STRING ed il simbolo *, come nel seguente esempio

DIM NOME AS STRING * 20

in cui la variabile NOME viene definita come stringa di 20 caratteri a lunghezza fissa.

Se si volesse dimensionare un array o una variabile semplice di un tipo definito dall'utente con TYPE..END TYPE, bisognerebbe usare necessariamente la clausola AS seguita dal nome del tipo di dati creato. Ecco un esempio che chiarisce quest'ultimo caso

```
TYPE PEZZO
        TIPO AS STRING * 10
        COLORE AS STRING * 1
        RIGAPOS AS STRING * 1
        COLPOS AS STRING * 1
END TYPE
DIM SCACCHIERA(1 TO 32) AS PEZZO
SCACCHIERA(1).TIPO = "RE"
SCACCHIERA(1).COLORE = "B"
SCACCHIERA(1).RIGAPOS = "E"
SCACCHIERA(1).COLPOS = "1"
END
```

Per quanto riguarda gli arrays, essi possono essere in Quick Basic, di due tipi: statici e dinamici; gli arrays sono statici quando il loro spazio e contenuto iniziale viene preparato al momento della compilazione, mentre sono dinamici quando sono preparati al momento dell'esecuzione della DIM corrispondente.

La dimensione complessiva che gli arrays, usati in un programma, possono raggiungere deve essere, insieme allo spazio usato per lo stack, minore di 64 K ma, se essi sono dinamici e viene specificata l'opzione /AH durante la compilazione o nel comando QB, essi possono tranquillamente superare tale limite.

Un array è statico nei seguenti casi

1. è stato specificato il metacomando $STATIC (anche se questo non è vincolante dato che, alcuni arrays, per come sono dichiarati, continuano ad essere dinamici, malgrado la sua presenza);

2. è stato dimensionato l'array usando come indici delle costanti anche se dichiarate con l'istruzione CONST;

mentre è dinamico in questi altri

1. è stato specificato un metacomando $DYNAMIC (che rende dinamici anche gli arrays dimensionati con indici costanti);

2. è stato dimensionato un array usando come indici delle variabili; dato che il contenuto di queste ultime non è conosciuto in partenza dal compilatore, questo non sa quanto spazio deve allocare per l'array, che viene dunque definito dinamico.

Con gli arrays dinamici, se eliminati tramite l'istruzione ERASE, è possibile recuperare la memoria occupata al fine di utilizzarla in altro modo, mentre non è così per quelli statici, in cui l'istruzione ERASE fa solo in modo di azzerare tutto l'array; in quest'ultimo caso, a differenza del

primo, non si deve eseguire un'altra frase DIM per riutilizzare gli arrays eliminati. Per potere modificare le dimensioni di un array dinamico, è disponibile l'istruzione **REDIM**.
Notare che con un'unica frase DIM è possibile dimensionare più elementi variabili separandoli con una virgola.

*** DO**

Questa istruzione, insieme a LOOP, WHILE, UNTIL ed EXIT DO, costituisce la maniera con cui, in Quick Basic, è possibile ripetere un blocco di istruzioni un numero indefinito di volte. La struttura del ciclo DO..LOOP può assumere forme differenti, dipendenti dall'uso che se ne vuole fare. La sintassi della prima forma prevede che la parola DO venga seguita da una delle due specifiche WHILE o UNTIL e da una espressione logica che, insieme, determinano le condizioni per cui il ciclo deve essere ripetuto o no. Le istruzione che fanno parte del ciclo, in questo caso, sono quelle che seguono l'espressione logica di controllo fino all'istruzione LOOP

```
DO {WHILE | UNTIL} espresso
    …
    [EXIT DO]
    …
LOOP
```

Secondo questa prima sintassi, il controllo dell'espressione logica avviene all'inizio del ciclo, prima dell'esecuzione della prima istruzione del blocco da ripetere; la seguente tabella riassume tutti i casi che si possono verificare in questa prima forma, consentendo un più facile uso dell'istruzione DO

	Opzione	
	WHILE	**UNTIL**
Esp. **FALSE**	Esce dal ciclo	Ripete il ciclo
Esp. **TRUE**	Ripete il ciclo	Esce dal ciclo

Con l'opzione WHILE, il ciclo viene ripetuto finché una certa condizione continua ad essere vera, mentre con l'opzione UNTIL, lo stesso viene ripetuto solo se la condizione resta falsa.

La seconda sintassi differisce dalla prima soltanto perché le opzioni WHILE ed UNTIL devono essere fatte seguire all'istruzione LOOP, cosa che determina il fatto che il controllo dell'espressione logica avviene alla fine del ciclo e non all'inizio; questo comporta che il contenuto del ciclo venga eseguito almeno una volta. Il seguente esempio mostra come è possibile scrivere con tale istruzione, alcune righe che controllino l'uso di un tasto, evidenziando le differenze tra le varie forme della struttura di controllo

1)
```
DO WHILE INKEY$ = “”
LOOP
```

2)
```
DO UNTIL INKEY$ <> “”
LOOP
```

In questo caso, il controllo dell'uso di un tasto viene fatto all'inizio del ciclo, anche se questo non avrebbe importanza ai fini del risultato, ed il ciclo non contiene alcuna istruzione da

ripetere dato che serve soltanto a fermare il programma temporaneamente; nella forma **1)** l'opzione WHILE specifica che **finché la funzione INKEY$ ritorna il valore stringa nulla**, e cioè finché nessun tasto è stato premuto, il ciclo deve essere ripetuto; invece, nella forma **2)** il ciclo viene ripetuto **fino a quando la funzione INKEY non ritorna un valore che è diverso da stringa nulla**, cioè fino a quando non sia stato premuto un tasto.
L'espressione logica può essere composta seguendo le stesse regole usate nell'istruzione IF e, se questa ritorna sempre il valore TRUE, il ciclo non ha mai termine; questo caso viene evitato dall'istruzione **EXIT DO** che, posta all'interno del ciclo ed eseguita all'interno di un blocco IF..END IF, permette di terminare lo stesso in determinate condizioni. Ad esempio, se si volesse interrompere un programma finché non si preme un tasto o, comunque, se passano 10 secondi anche se non è premuto alcun tasto, si può usare l'istruzione EXIT DO come nel seguente programma

```
K!=TIMER
DO WHILE INKEY$=""
        IF TIMER-K!>10 THEN
                EXIT DO
        END IF
LOOP
```

La struttura DO..LOOP può essere nidificata, cioè può essere contenuta all'interno di un altro ciclo DO..LOOP ed in questo caso l'istruzione EXIT DO si riferisce al ciclo in cui la stessa è posta.
È meglio usare la struttura DO..LOOP nei casi in cui si dovrebbe usare l'istruzione GOTO dato che così il programma assume una forma più lineare , esteticamente migliore ed, in caso di errori, è più facilmente ispezionabile; ad esempio, le seguenti righe di programma

```
CLS
K=0
NEWINP:
        LOCATE 10,10
        INPUT "Valore : ", K
IF K<70 OR K>120 THEN GOTO NEWINP
```

possono essere sostituite da queste altre

```
CLS
K=0
DO
        LOCATE 10,10
        INPUT "Valore : ", K
LOOP WHILE K<70 OR K>120
```

che usano la struttura di controllo DO..LOOP.

*** DOUBLE**

Vedere AS, COMMON, DECLARE, DEF FN, DIM, FUNCTION, SHARED, STATIC, SUB, TYPE

*** DRAW**

È un'istruzione usata in grafica per disegnare usando un gruppo di comandi inseriti in una stringa che costituiscono una sorta di linguaggio dedicato alla realizzazione di figure. La sintassi prevede, obbligatoriamente, che la parola DRAW venga seguita da un'espressione stringa contenente varie combinazioni dei comandi riconosciuti, elencati di seguito:

U [n]	sposta in alto il cursore grafico di n unità;
D [n]	sposta in basso il cursore grafico di n unità;
L [n]	sposta a sinistra il cursore grafico di n unità;
R [n]	sposta a destra il cursore grafico di n unità;
E [n]	sposta in alto a destra il cursore grafico di n unità;
F [n]	sposta in basso a destra il cursore grafico di n unità;
H [n]	sposta in alto a sinistra il cursore grafico di n unità;
G [n]	sposta in basso a sinistra il cursore grafico di n unità;
M x,y	sposta il cursore grafico nella posizione assoluta avente coordinate x,y;
M {+|-}x,{+|-}y	sposta il cursore grafico nella posizione relativa rispetto a quella attuale dello stesso, variando le coordinate x e y con le quantità e i segni indicati;
B	è un prefisso da mettere prima di ogni comando già descritto che permette di effettuare lo spostamento senza disegnare alcunché sullo schermo;
N	è un prefisso che messo prima di ogni comando di spostamento permette di disegnare senza muovere il cursore dalla posizione iniziale;
C n	con questo comando è possibile modificare il colore di primo piano, quello con cui vengono tracciati i segmenti; se questo comando può essere usato e il numero che si può specificare dopo di esso, dipende dalla scheda grafica e dal monitor usato; vedere l'istruzione COLOR per ulteriori chiarimenti;
P col,bcol	è il comando equivalente al PAINT; provvede a colorare l'interno di una figura con un colore prescelto; col è il colore da usare e bcol è il colore del bordo della figura stessa; non è possibile usare motivi personalizzati per il riempimento come nella istruzione DRAW;
S n	serve a stabilire il fattore di scala; le unità trattate nei comandi di spostamento del cursore grafico sono equivalenti al numero n del comando S; per default, questo valore è 4;
A n	è uno dei due comandi di rotazione; permette di ruotare il segmento da tracciare di un valore equivalente a n x 90 gradi; n è comunque compreso tra 0 e 3 ed è un intero;
TA n	è l'altro comando di rotazione; è più flessibile del precedente in quanto il valore n, che è compreso tra 0 e 360, è la misura in gradi della rotazione;
X	è il comando per l'esecuzione di una stringa che contiene comandi riconosciuti dall'istruzione DRAW.

Nei comandi di spostamento del cursore grafico nelle varie direzioni, se nessun prefisso è specificato, viene disegnato il segmento corrispondente e la posizione del cursore viene aggiornata; se non viene specificato il numero n, esso è 1 per default.

Ecco un esempio che mostra l'uso della maggior parte dei comandi che si possono usare con l'istruzione DRAW; si tratta di alcune linee che simulano in grafica la caduta a vite di un aereo dal quale si avvistano due palazzi stilizzati

```
SCREEN 1
A$="H20L40D100F20U100NH20R40D100L40BH20NL150BR65R100"
S=1
DX=50
K=.1
DO WHILE 1
    FOR T=3.1415 TO -3.1415 STEP –K
            S=S + .005
            IF S<6.3 THEN
                    PX=INT(SIN(T) * 30) + 100
                    PY=INT(COS(T) * 30) + 30
            ELSE
                    PY=PY-4
            END IF
            DRAW "C3S" + STR$(INT(S))
            DRAW "BM" + STR$(PX) + "," + STR$(PY) + A$
            DRAW "BM" + STR$(PX+INT(DX)) + "," + STR$(PY) + A$
            IF PY<-150 THEN
                    PRINT "CRASH"
                    END
            END IF
            DRAW "C0"
            DRAW  "BM" + STR$(PX) + "," + STR$(PY) + A$
            DRAW "BM" + STR$(PX+INT(DX)) + "," + STR$(PY) + A$
            DX = DX + .2
    NEXT T
LOOP
```

Notare come i valori numerici variabili, se usati, devono essere interi e convertiti con la funzione STR$.

Quando viene usato il comando X che esegue dei comandi posti in una stringa ausiliaria, si devono rispettare alcune regole imposte dal Quick Basic; dopo il comando X non deve essere posta la variabile stringa ma ciò che ritorna la funzione VARPTR$ appositamente definita a tale scopo; il comando X va quindi usato come nel seguente esempio

```
CONST A$ = "S1C3U100R100D100L100BE5P2,3"
CONST B$ = "S1C3F100L200E100BD5P2,3"
DEFINT A-Z
SCREEN 1
RANDOMIZE TIMER
DO WHILE 1
      PX = RND * 300
      PY = RND * 200
      DRAW "BM" + STR$(PX) + "," + STR$(PY)
      K = RND * 100
      IF K<40 THEN
```

```
            W$ = A$
        ELSEIF K<99 THEN
            W$ = B$
        ELSE
            CLS 0
        END IF
        DRAW "X" + VARPTR$(W$)
    LOOP
```

Le stringhe A$ e B$ sono definite come costanti per una questione di velocità e di comodità ma possono essere delle semplici variabili; è possibile usare elementi di array, ma se questi sono dinamici, solo se si ci trova nell'ambiente Quick Basic; non è possibile inoltre, che una stringa eseguita con il comando X richiami sé stessa, come nel seguente esempio

```
SCREEN 1
K$ = "M100,100X" + VARPTR$(K$)
DRAW "X" + K$
```

in cui la variabile K$ contiene un riferimento a sé stessa; in questo caso, mentre nella versione 4.5 del Quick Basic, viene comunque evidenziato un errore per superamento delle capacità dello stack, nella versione 4.0, se si ci trova nell'ambiente, viene evidenziato un errore non molto comune: *DOS memory arena error* che informa di un conflitto avvenuto nella memoria da parte del DOS; è un errore critico che fa immediatamente tornare il Quick Basic al Sistema Operativo; sempre nella stessa versione, se compilato con BC, l'errore manifestato dal programma in questione è quello usuale di Overflow dello Stack.

*** ELSE**

Vedere IF

*** ELSEIF**

Vedere IF

*** END**

È l'istruzione usata per terminare un programma in Quick Basic; essa chiude tutti i file e, se il programma era stato eseguito nell'ambiente, permette di tornare a quest'ultimo; se il programma era già stato compilato e registrato in un file EXE, permette di tornare al DOS.

Non è un'istruzione obbligatoria, in quanto il Quick Basic ne inserisce una automaticamente alla fine del programma, ma se prima della fine dello stesso è presente una subroutine richiamabile con l'istruzione GOSUB, allora questa deve essere separata dal modulo che la richiama, con l'istruzione END; in caso contrario la subroutine sarebbe eseguita e, appena incontrata l'istruzione RETURN, verrebbe emesso l'errore relativo al fatto che non esisteva una GOSUB, come nel caso seguente

```
CLS
INPUT "VALORI : ",X,Y
GOSUB PIU
GOSUB MENO
```

```
PIU:
  PRINT X+Y
  RETURN
MENO:
  PRINT X-Y
  RETURN
```

che, per evitare errori, deve essere corretto ponendo l'istruzione END dopo la seconda istruzione GOSUB

```
…
GOSUB PIU
GOSUB MENO
END
…
```

*** END DEF**
Vedere DEF FN

*** END FUNCTION**
Vedere FUNCTION

*** END IF**
Vedere IF

*** END SELECT**
Vedere SELECT

*** END SUB**
Vedere SUB

*** END TYPE**
Vedere TYPE

*** ENVIRON**
È l'istruzione complementare alla funzione ENVIRON$; permette di modificare la tabella delle variabili di ambiente del DOS. La sua sintassi è la seguente

ENVIRON espstringa

in cui 'espstringa' è una espressione di tipo stringa in cui contenuto può essere uno dei seguenti

"nomevar=dato"
oppure
"nomevar dato"

in cui 'nomevar' è il nome della variabile di ambiente interessata e 'dato' è il valore che essa deve assumere. Il segno di uguale può essere sostituito dallo spazio senza alterare il

funzionamento dell'istruzione. Questa istruzione, se la variabile di ambiente non era stata preparata in DOS con il comando SET, emette una segnalazione di errore (Memoria non disponibile), per indicare che non esiste spazio nella tabella delle variabili di ambiente; dopo che lo spazio è preparato con il comando SET, il contenuto della variabile può essere modificato ponendo attenzione che la lunghezza in caratteri del suo nuovo contenuto non ecceda quello esistente; si può sfruttare lo spazio messo a disposizione dopo aver cancellato una variabile per definirne un'altra; per eliminare una variabile di ambiente dalla tabella si deve fare seguire il simbolo ; al segno di uguale oppure si può chiudere la stringa subito dopo lo stesso segno; in questo modo la variabile di ambiente indicata viene cancellata e lo spazio diventa disponibile per un'altra, come nell'esempio seguente

In DOS eseguire SET VAR=************
ed in Quick Basic

```
ENVIRON “VAR=;”          ‘ Elimina la variabile VAR
ENVIRON “VAR1=55”        ‘ Prepara la variabile VAR1
ENVIRON “VAR2=C:\”       ‘ Prepara la variabile VAR2
```

Porre particolare attenzione al fatto che le variabili di ambiente preparate in questo modo **non vengono conservate al rientro in DOS** in quanto viene ristabilita la tabella originaria che il DOS aveva al momento della partenza del Quick Basic; questa istruzione può quindi servire solo per passare, tramite l'istruzione SHELL, delle variabili a programmi che le interpreteranno, ma le stesse non saranno conservate. Ad esempio, ecco le operazioni da fare per usare un programma in Quick Basic che, sotto DOS 3.30, passa un parametro ad un file batch per la formattazione di un disco (cosa che sarebbe possibile fare anche con il solo comando SHELL)

Eseguire in DOS il comando SET DR=**
Con EDLIN preparare in DOS il seguente file batch

```
@ECHO OFF
ECHO FORMATTAZIONE DRIVE %DR% …
FORMAT %DR%
```

Scrivere in Quick Basic il programma

```
‘ (Main Module)
CLS
INPUT “DRIVE “,DR$
ENVIRON “DR=”+DR$+”:”
SHELL “FMT.BAT”
PRINT
PRINT “FINE OPERAZIONE.”
END
```

Per potere controllare il contenuto di una variabile d'ambiente preparato con il comando ENVIRON, in Quick Basic, riferirsi alla funzione ENVIRON$.

*** EQV**

L’operatore logico EQV esegue l’operazione di confronto bit a bit, tra due valori interi o interi lunghi, bit per bit, secondo le regole della tavola seguente:

1^ Operando	2^ Operando	Risultato
0 (FALSE)	0 (FALSE)	**1 (TRUE)**
0 (FALSE)	**1 (TRUE)**	0 (FALSE)
1 (TRUE)	0 (FALSE)	0 (FALSE)
1 (TRUE)	**1 (TRUE)**	**1 (TRUE)**

L’operatore può essere usato in qualsiasi espressione logico-aritmetica ma, come gli altri operatori logici, possiede il più basso livello di priorità.

L’operatore EQV ritorna il valore vero solo se i due valori confrontati sono eguali: perciò i due esempi seguenti forniscono lo stesso risultato

```
' (Main Module EQV1.BAS)
CLS
INPUT A,B
IF A=B THEN
        PRINT "I DUE VALORI SONO UGUALI."
ELSE
        PRINT "I DUE VALORI SONO DIVERSI."
END IF
END

' (Main Module EQV2.BAS)
CONST FALSO=0, VERO=NOT FALSO
CLS
INPUT A,B
IF (A EQV B)=VERO THEN
        PRINT "I DUE VALORI SONO UGUALI."
ELSE
        PRINT "I DUE VALORI SONO DIVERSI."
END IF
END
```

Notare le parentesi che racchiudono l’espressione A EQV B; esse sono necessarie perché, per la priorità degli operatori, l’espressione sarebbe interpretata come A EQV (B=VERO) in cui il segno di eguale inteso come operatore di relazione, verrebbe calcolato per primo.

*** ERASE**

L’istruzione ERASE è usata con gli arrays; la sua sintassi è la seguente

ERASE nomearray [,nomearray][…]

in cui ‘nomearray’ sono i nomi degli arrays su cui l’istruzione deve agire; se l’array è di tipo statico, l’istruzione ERASE provvede ad azzerare tutti gli elementi numerici, porre eguale a

stringa nulla gli elementi alfanumerici inclusi quelli appartenenti ad un record definito con TYPE..END TYPE; invece, se l'array è di tipo dinamico, l'istruzione ERASE libera la memoria occupata dall'array e la restituisce al DOS per poterla usare in seguito; anche in questo caso tutti i dati dell'array sono persi.

Ad esempio, nel seguente caso

```
N=10
DIM A(1 TO 10), B(1 TO N)
ERASE A, B
```

l'array A viene semplicemente azzerato, dato che è statico, mentre lo spazio occupato dall'array B, viene liberato perché di tipo dinamico; ponendo il metacomando $DYNAMIC, come nel prossimo esempio

```
' $DYNAMIC
N=10
DIM A(1 TO 10), B(1 TO N)
ERASE A, B
```

anche lo spazio occupato dall'array A viene liberato in quanto anch'esso è definito dinamico; ricordare che gli arrays dichiarati implicitamente, sono sempre di tipo statico e quindi nel seguente esempio, l'array C viene semplicemente azzerato, anche se esiste il metacomando $DYNAMIC

```
' $DYNAMIC
CLS
C(5) = 15
ERASE C
PRINT C(5)
```

Dopo un'istruzione ERASE è possibile utilizzare nuovamente l'istruzione DIM o la REDIM per preparare gli arrays eliminati, ma solo se questi sono di tipo dinamico; se sono statici vale sempre la prima frase DIM eseguita per ognuno di essi.

*** ERROR**

Con questa istruzione è possibile simulare l'avvenire di un errore, sia standard che di tipo utente; la sua sintassi prevede che sia seguita da un valore intero, cioè

ERROR esprintera

compreso tra 0 e 255; nelle versioni 4.0 e 4.5 di Quick Basic, gli errori con codice superiore al 76 non sono definiti, ma è così anche per qualche codice che ha numero inferiore; il messaggio di errore che Quick Basic emette per questi errori è Unprintable Error (Errore non visualizzabile); essi possono essere utilizzati dall'utente che li può gestire prevedendoli nella routine di controllo degli errori (vedere ON ERROR GOTO); per una lista completa degli errori, vedere l'Appendice relativa.

Ecco un esempio che illustra l'uso dell'istruzione ERROR, nella simulazione di un errore già definito in Quick Basic

```
ON ERROR GOTO ERTRAP
CLS
NEWINP:
        PRINT
        INPUT "VALORE ", A
        IF A<5 OR A>10 THEN
                ERROR 5
        END IF
        PRINT A
        END
ERTRAP:
        IF ERR=5 THEN
                PRINT "VALORE NON COMPRESO NEI LIMITI"
                RESUME NEWIMP
        ELSE
                ON ERROR GOTO 0
        END IF
```

Si può usare uno dei codici che il Quick Basic mette a disposizione dell'utente, ad esempio il 255, per realizzare il seguente esempio

```
ON ERROR GOTO ERTRAP
CLS
FOR T=-5 TO 5
                PRINT "RADICE QUADRATA DI ";T;"=";
                IF T<0 THEN ERROR 255
                PRINT SQR(T)
NXTVAL:
        NEXT T
        END
ERTRAP:
        IF ERR=255 THEN
                PRINT "NON DEFINITA"
                RESUME NXTVAL
        ELSE
                ON ERROR GOTO 0
        END IF
```

*** EXIT DEF**

Vedere DEF

*** EXIT DO**

Vedere DO

*** EXIT FOR**

Vedere FOR

*** EXIT FUNCTION**
Vedere FUNCTION

*** EXIT SUB**
Vedere SUB

*** FIELD**
Con questa istruzione è possibile specificare i nomi e le lunghezze dei campi che costituiscono il record di un file random; il totale di queste lunghezze non deve superare il valore della lunghezza del record specificato nell'istruzione OPEN; se questo valore manca, esso è, per default, uguale a 128.
La sintassi dell'istruzione FIELD prevede l'immissione del numero del file aperto con una precedente istruzione OPEN, seguito dalla lunghezza e dal nome di ogni campo costituente il record, come espresso qui di seguito

FIELD [**#**]**numfile, lungcampo AS nome campo** [**, …**]

in cui numfile, eventualmente preceduto dal simbolo opzionale #, è il numero del file corrispondente a quello indicato nell'istruzione OPEN corrispondente; segue poi la lista delle definizione dei campi con la lunghezza ed il nome degli stessi separati dalla clausola AS; la lunghezza di un campo può essere, al massimo, eguale a 32767 caratteri che corrisponde alla lunghezza massima di una stringa; infatti i campi sono tutti definiti come stringhe e i loro nomi rispettano le regole imposte per la definizione dei nomi delle variabili alfanumeriche. Notare che, se una variabile alfanumerica è definita come campo in un'istruzione FIELD di un record di un file random, essa non può essere usata nel programma come normale variabile stringa in quanto è definita nell'area buffer del file e non nell'area appositamente riservata alle variabili; un uso improprio delle variabili definite nelle istruzioni FIELD, potrebbe portare ad una cattiva lettura e scrittura dei dati sul file corrispondente alla FIELD considerata. I dati numerici possono essere scritti e letti sul file, servendosi delle funzioni MK..$ e CV.. esistenti per ogni tipo di dato numerico che consentono di usare, nell'istruzione FIELD, stringhe con una lunghezza determinata per i numeri; le lunghezze sono le seguenti

Tipo Numerico	Lunghezza Stringa
Intero	2
Intero Lungo	4
Singola prec.	4
Doppia prec.	8

Ad esempio, per definire un record di un file random contenente alcuni dati per alcune persone, si ricorre al programma costituito dalle seguenti linee

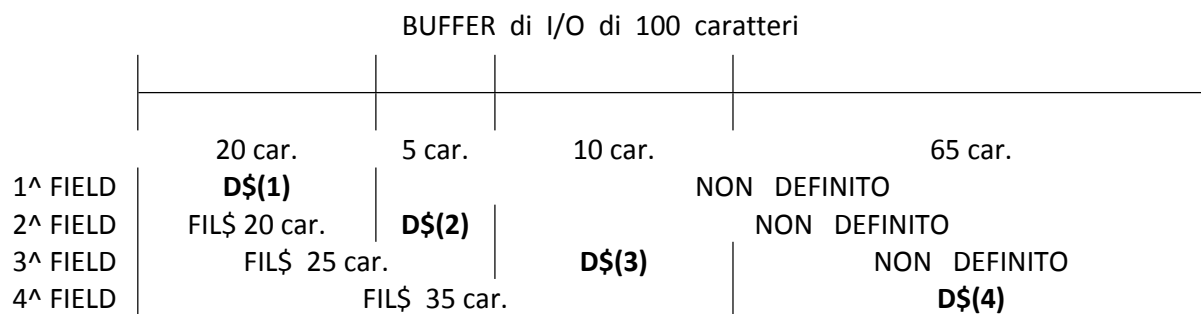
```
OPEN "R",#1,"PROVA.DAT",48
FIELD #1,40 AS NOM$, 2 AS ETA$, 2 AS LIV$, 4 AS STIP$
```

in cui il campo del nome è definito come stringa di 40 caratteri, l'età come stringa di 2 caratteri perché si useranno le funzioni di conversione per gli interi, il livello come stringa di 2 caratteri

come nel caso precedente e lo stipendio come stringa di 4 caratteri perché si useranno le funzioni di conversione degli interi lunghi; la lunghezza totale del record, 48 caratteri, è definita come somma delle lunghezze di tutti i campi.

Se sono definite più istruzioni FIELD per un file, esse valgono tutte contemporaneamente, considerando che a uguali posizioni del buffer di I/O è possibile far corrispondere diverse variabili di diverse istruzioni FIELD. È un modo di funzionamento utile quando i campi devono essere assegnati ad elementi di un array di tipo alfanumerico; in questo caso sarebbe particolarmente tedioso esprimere la FIELD con una sequenza di nomi di elementi di array, come nel caso seguente

```
DIM D$(1 TO 10)
OPEN "R",#1,"PROVA.DAT",100
FIELD #1, 20 AS D$(1), 5 AS D$(2), 10 AS D$(3), 65 AS D$(4)
```

mentre è più comodo definire tutti i campi considerando che diverse istruzioni FIELD agiscono sullo stesso buffer così da potere avere il seguente schema

che è semplicemente realizzabile con un ciclo di tipo FOR..NEXT e delle frasi DATA, come nel seguente esempio

```
DIM D$(1 TO 4)
OPEN "R",#1,"PROVA.DAT",100
RES=0
FOR T=1 TO 4
        READ L
        FIELD #1, RES AS FIL$, L AS D$(T)
        RES=RES+1
NEXT T
…
DATA 20,5,10,65
```

Questo metodo è valido, sia che l'array sia composto da 4 elementi, come nell'esempio, che da 100 o più elementi così da consentire di usare molti campi in un record molto semplicemente; in ogni caso, il totale delle lunghezze dei campi non può superare i 32767 caratteri.

Comunque, per una più razionale definizione dei campi di un record di un file random, Quick Basic mette a disposizione la struttura TYPE..END TYPE che permette di definire il record come tipo di dati utente da sfruttare, in blocco, per l'I/O da file.

*** FILE**

Il comando FILE provvede ad elencare su video il contenuto della directory di lavoro del disco di default. Viene presentata una testata che indica, appunto, quest'ultime informazioni, l'elenco di tutti i file completi di nome ed estensione (come nel comando DIR del DOS con l'opzione /P), e per ultima, l'area libera si disco espressa in bytes. È un comando poco utilizzato nei programmi utente dato che il formato con cui presenta l'elenco dei file, è fisso ed immutabile e che i dati forniti sono limitati; manca infatti, per ogni file, la lunghezza, la data e l'ora di creazione, gli attributi. La sintassi

FILE [**specfile**]

prevede che all'istruzione segua una espressione stringa con cui è possibile specificare un percorso di ricerca diverso dall'attuale ed una maschera che provveda a selezionare un insieme di file specificato; per fare ciò è possibile usare i caratteri jolly caratteristici del DOS e cioè * e ?. Notare che, anche se viene specificato un percorso di ricerca diverso dall'attuale, nella testata viene sempre proposto, forse per errore, il percorso completo di lavoro; ad esempio, se si ci trovasse nella directory di lavoro QB40 della principale e si volesse elencare il contenuto della directory QB45 della principale, si dovrebbe dare il seguente comando nella finestra Immediate

FILE "\QB45*.*"

e si otterrebbe la lista dei file contenuti nella directory \QB45, ma con la testata \QB40, cioè quella di lavoro.
Per ottenere una migliore gestione nei programmi del contenuto delle directory, è meglio affidarsi a delle routines scritte in altri linguaggi che si interfaccino con il DOS per l'estrazione dei dati richiesti.

*** FOR**

L'istruzione FOR, insieme a TO, STEP, EXIT FOR e NEXT, costituiscono il modo con cui, in Quick Basic, è possibile definire dei cicli che si ripetano un determinato numero di volte. La sintassi completa della struttura FOR, è la seguente

FOR nomevar=esprenum1 TO esprnum2 [**STEP esprnum3**]
  …
  [**EXIT FOR**]
  …
NEXT [[**nomevar**][**, …**]]

Le istruzioni comprese tra l'istruzione FOR e l'istruzione NEXT corrispondente, vengono ripetuto un numero di volte dipendente dai valori assunti dalle espressioni numeriche **esprnum1**, **esprnum2**, e, se esiste, dalla **esprnum3**. All'inizio del ciclo, il valore della prima espressione viene assegnato, come valore iniziale, alla variabile numerica specificata dopo l'istruzione FOR, detta anche variabile di controllo del ciclo; l'esecuzione del ciclo continua fino all'istruzione NEXT che lo chiude e a questo punto avvengono le seguenti operazioni

- il valore della esprnum3, se presente, viene aggiunto alla variabile di controllo del ciclo; se la esprnum3 manca (non è usata l'istruzione STEP) questo valore è assunto uguale a 1;

- il contenuto della variabile di controllo viene confrontato con il valore della esprnum2 e di conseguenza, il ciclo può essere ripetuto o no; la tabella successiva riassume tutti i casi che si possono verificare

esprnum1=esprenum2	esprnum3>0	**ciclo ripetuto 1 volta**
esprnum1=esprenum2	esprnum3<0	**ciclo ripetuto 1 volta**
esprnum1=esprenum2	esprnum3=0	**ciclo infinito**
esprnum1>esprenum2	esprnum3>0	**ciclo non eseguito**
esprnum1>esprenum2	esprnum3<0	**ciclo eseguito k volte**
esprnum1>esprenum2	esprnum3=0	**ciclo non eseguito**
esprnum1<esprenum2	esprnum3>0	**ciclo eseguito k volte**
esprnum1<esprenum2	esprnum3<0	**ciclo non eseguito**
esprnum1<esprenum2	esprnum3=0	**ciclo infinito**

in cui k è eguale al risultato della seguente espressione

INT(ABS(esprnum2-esprnum1)*ABS(1/esprnum3))+1

In qualsiasi momento comunque, tramite l'istruzione EXIT FOR, è possibile terminare un qualsiasi ciclo FOR..NEXT anche se il contenuto della variabile di controllo non lo consentisse.
Il nome della variabile di controllo può essere fatto seguire all'istruzione NEXT, ma in sua mancanza, l'istruzione NEXT si riferirà sempre al ciclo aperto per ultimo; è possibile infatti, nidificare i cicli FOR..NEXT, come per i cicli DO..LOOP, inserendone uno nell'altro; per fare ciò è obbligatorio mantenere diverse le variabili di controllo dei cicli e chiudere il ciclo più interno prima di quello più esterno. Attenzione quindi se si dovesse usare il nome della variabile di controllo nell'istruzione NEXT. Ecco un esempio di uso scorretto dei cicli FOR..NEXT nidificati

```
FOR K=1 TO 10 STEP 2
   FOR T=1 TO 10
          ...
   NEXT K
NEXT T
```

infatti le due frasi NEXT dovrebbero essere invertite permettendo la chiusura del ciclo controllato da T prima di quello controllato da K; ciò si ottiene anche con un'unica frase del tipo

```
NEXT K, T
```

che provvede a chiudere prima il ciclo K e poi quello T. I valori espressi come inizio, fine e passo del ciclo possono essere delle espressioni numeriche qualsiasi, e la loro precisione, come quella della variabile di controllo, può essere una qualsiasi tra quelle consentite dal linguaggio; come esempio di uso del ciclo FOR..NEXT, ecco un programma che provvede a controllare se esistono stringhe palindrome in un vettore alfanumerico

```
CLS
DIM A$(1 TO 5)
```

```
A$(1) = "NOI"
A$(2) = "LORO"
A$(3) = "ESSE"
A$(4) = "VOI"
A$(5) = "IO"
FOR EL=1 TO 5
      E$=""
      FOR CH=LEN(A$(EL)) TO 1 STEP -1
            E$=E$+MID$(A$(EL), CH, 1)
      NEXT CH
      IF E$=A$(EL) THEN
            COLOR 15
            PRINT A$(EL); " E' PALINDROMA."
      ELSE
            COLOR 7
            PRINT A$(EL); " NON E' PALINDROMA."
      END IF
NEXT EL
```

*** FUNCTION**

È una parola chiave della frase di dichiarazione usata per definire una Function in Quick Basic. In modo del tutto similare alla parola SUB, è seguita dal nome di una Function, nome che rispetta le regole usate per la costruzione dei nomi delle variabili; il nome stesso è fatto seguire da uno dei simboli caratteristici (%, &, !, #, $) che definiscono il tipo di dato che la funzione deve ritornare al programma chiamante; se utilizzati, tra parentesi tonde vengono specificati i parametri richiesti dalla funzione, con il loro nome e tipo tramite i caratteri già visti o la clausola AS seguita dalle parole chiave INTEGER, LONG, SINGLE, DOUBLE o STRING; se bisogna passare un array come parametro, è sufficiente specificare dopo il nome la coppia di parentesi tonde aperte-chiuse senza obbligo di scrivere il numero di dimensioni dell'array come in versioni precedenti del linguaggio. La sintassi completa della frase di dichiarazione è la seguente

FUNCTION nomefunction[**%** | **&** | **!** | **#** | **$**] [**(listaparametri)**] [**STATIC**]

in cui (listaparametri) è la lista dei parametri usati dalla Function, ed ha la seguente sintassi

nomevar[**( )**] [**AS** {**INTEGER**|**LONG**|**SINGLE**|**DOUBLE**|**STRING**}][**,…**]

La parola STATIC che può essere aggiunta alla fine della dichiarazione, fa in modo che il Quick Basic conservi il valore delle variabili locali usate, tra una chiamata e l'altra; esse infatti, normalmente sono cancellate al termine dell'esecuzione della Function e sono create nuovamente ad una sua nuova chiamata; questa operazione fa perdere un po' di tempo durante l'esecuzione dei un programma che contenga molte chiamate a Function, e quindi l'uso della parola STATIC velocizza, in maniera limitata, le Functions; non è consigliabile usare questa frase con funzioni ricorsive (che chiamano sé stesse).

Per creare una Function è possibile agire tramite il menu Edit (New FUNCTION) nel qual caso la frase di dichiarazione della FUNCTION viene inserita automaticamente dal Quick Basic; se invece viene specificata tale parola durante l'editing di un programma, con il tasto Return, la

nuova Function viene creata insieme alla frase END FUNCTION che la delimita; il valore della Function viene passato tramite il nome della stessa; è possibile uscire dalla Function prima che la stessa abbia termine, tramite l'istruzione EXIT FUNCTION.

Ecco un esempio di utilizzo di una Function che determina lo stato delle stampanti parallele collegate al sistema

```
' (Main Module)
DECLARE FUNCTION LPTSTATUS$(LPT%)
DEFINT A-Z
CLS
FOR T=1 TO 3
        PRINT "LPT";T;": "; LPTSTATUS(T)
NEXT T
FUNCTION LPTSTATUS$(LPT%) STATIC
        DEFINT A-Z
        IF LPT<1 OR LPT>3 THEN
                LPTSTATUS = "Not available"
                EXIT FUNCTION
        END IF
        DEF SEG = &H40
        LPTAD = PEEK(6 + LPT * 2) + PEEK(7 + LPT * 2) * 256
        DEF SEG
        IF LPTAD=0 THEN
                LPTSTATUS = "Not installed"
                EXIT FUNCTION
        END IF
        ST = INP(LPTAD+1)
        SELECT CASE ST
                CASE &H57
                        LPTSTATUS = "Off-Line"
                CASE &H77
                        LPTSTATUS = "Out of Paper"
                CASE &H87
                        LPTSTATUS = "Lpt Off"
                CASE &HDF
                        LPTSTATUS = "On-Line"
                CASE &HFF
                        LPTSTATUS = "Reset"
                CASE ELSE
                        LPTSTATUS = "Undefined"
        END SELECT
END FUNCTION
```

La funzione LPTSTATUS restituisce una stringa contenente lo stato della stampante collegata al sistema; tale stampante può essere la LPT1, LPT2 o LPT3; il numero della stampante di cui si vuole conoscere lo stato va passato come parametro alla funzione (LPT%); la funzione restituisce una stringa secondo la seguente tabella

Not installed	Scheda di interfaccia non presente
Reset	Stampante attualmente in reset; questa condizione si ottiene quando la stampante è collegata e si richiede lo status proprio quando la si attiva
On-Line	Stampante attivata e on-line
Off-Line	Stampante attivata e off-line
Out of Paper	Stampante attivata e off-line perché manca la carta
Lpt Off	La scheda di interfaccia è presente ma la stampante manca o è spenta
Undefined	Lo stato della stampante non è definito
Not available	Il numero della stampante specificato non è valido; le stampanti possono essere la LPT1, LPT2 o LPT3 (l'argomento passato, quindi, può assumere solo i valori da 1 a 3, estremi inclusi)

*** GET**
(Istruzione di I/O)

L'istruzione GET di Quick Basic, usata per l'I/O dei file random e dei file binari, ha la seguente sintassi

GET [**#**]**numfile**[**,**[{**numrecord** | **byte**}]**, variabile**]

in cui 'numfile' è il numero di file associato all'istruzione OPEN corrispondente. Il secondo parametro, quando presente, nel caso dei file random, è il numero del record da leggere, e nel caso dei file binari, è la posizione all'interno dello stesso da cui cominciare a leggere. Il numero del record, per i file random, può essere compreso tra 1 e 2147483647 mentre la posizione che è possibile specificare, può essere compresa nei limiti della lunghezza del file aperto. Se questo parametro manca, i valori assunti sono quelli correnti; nel caso del file random, all'apertura il numero di record iniziale è 1 e poi, in seguito a letture o scritture, questo viene aggiornato; lo stesso avviene nel caso di file binari per quanto riguarda la posizione di lettura.
L'ultimo parametro, la variabile di input, è utilizzata in due casi: se viene aperto un file binario, è una variabile stringa e rappresenta il buffer di lettura e la sua lunghezza, dichiarata in precedenza, rappresenta il numero di bytes da leggere; per un file random, essa è invece una variabile di tipo definito dall'utente in quanto rappresenta il record usato dal file stesso; in questo caso non si deve usare l'istruzione FIELD all'apertura del file random.
Ecco un esempio di lettura e ricerca su un file random in cui si utilizza la maniera tradizionale per definire il record

```
OPEN "R",#1,"PROVA",32
FIELD #1,15 AS COGNOME$,15 AS NOME$,2 AS ETA$
NR&=LOF(1)
DO WHILE NR&>0
        GET #1
        IF RTRIM$(COGNOME$)="GIULIANA" THEN
                PRINT COGNOME$, NOME$, CVI(ETA$)
                CLOSE #1
                END
        END IF
        NR& = NR&-1
LOOP
```

```
PRINT "RECORD NON TROVATO"
CLOSE #1
END
```

Notare come, in questo caso, la ricerca avvenga dal primo record all'ultimo dato che si è omessa la sua specifica nell'istruzione GET; per potere eseguire la ricerca dall'ultimo fino al primo record, in maniera inversa rispetto alla precedente, dato che il contatore NR& procede in questa direzione, è sufficiente specificarlo nell'istruzione GET nel seguente modo

```
GET #1, NR&
```

L'istruzione FIELD determina la necessità di specificare la lunghezza del record a priori nell'istruzione OPEN e di usare la funzione CVI per ricavare il dato ETA definito come un numero intero codificato nel file random; questi obblighi non sono necessari se il programma precedente viene scritto nel seguente modo

```
TYPE RECPROVA
    COGNOME AS STRING *15
    NOME AS STRING * 15
    ETA AS INTEGER
END TYPE
DIM PREC AS RECPROVA
OPEN "R",#1,"PROVA",LEN(PREC)
NR&=LOF(1)
DO WHILE NR&>0
    GET #1,,PREC
    IF RTRIM$(COGNOME$)="GIULIANA" THEN
        PRINT PREC.COGNOME, PREC.NOME, PREC.ETA
        CLOSE #1
        END
    END IF
    NR& = NR&-1
LOOP
PRINT "RECORD NON TROVATO"
CLOSE #1
END
```

Notare le seguenti differenze

* non è stata utilizzata l'istruzione FIELD in quanto è stata definita la variabile PREC di tipo utente RECPROVA contenente tutti i dati necessari;

* la lunghezza del record è stato definito nell'istruzione OPEN in modo parametrico come lunghezza della variabile PREC; in tal modo variando il numero o le lunghezze dei campi definiti nel tipo record RECPROVA, non bisogna calcolare la sua lunghezza che è determinata automaticamente dalla funzione LEN;

* la variabile ETA è definita come intera e quindi è possibile usarla senza fare uso di funzioni di conversione; ciò vale anche se si fossero definiti dei campi in altre precisioni numeriche;

* nell'istruzione GET compaiono due virgole per indicare che il numero di record non è specificato ma esiste il nome della variabile PREC per indicare che i dati, dopo essere stati letti, devono essere depositati all'interno della stessa e divisi nei vari campi;

* i campi stessi sono definiti come elementi del record PREC e vanno utilizzati quindi con la scrittura PREC.COGNOME, PREC.NOME e PREC.ETA senza dimenticare il punto ed omettendo qualsiasi carattere che ne definisca il tipo.

Specificando l'opzione SHARED nella frase DIM che prepara la variabile PREC, è possibile condividere il record così definito con tutte le Functions e le Subs eventualmente usate nel programma.

Per quanto riguarda i file binari, ecco un esempio che utilizza l'istruzione GET per tali file; questo programma dimostrativo esegue la copia di un file se quest'ultimo è più corto di 32767 caratteri

```
CLS
INPUT "NOME DEL FILE DA COPIARE : ", OLDFILE$
INPUT "NOME NUOVO FILE : ", NEWFILE$
OPEN "B",#1,OLDFILE$
LF=LOF(#1)
IF LF>32767 THEN
  PRINT "FILE TROPPO LUNGO."
  CLOSE #1
  END
END IF
OPEN "B",#2,NEWFILE$
S$=SPACE$(LF)
GET #1,,S$
PUT #2,,S$
CLOSE #1, #2
END
```

In questo caso, il contenuto del file viene letto, a partire dal primo byte in quanto manca tale indicazione, per una lunghezza che dipende dalla lunghezza della variabile S$ usata nell'istruzione GET; dato che tale lunghezza corrisponde a quella del file stesso, tutto il file è quindi depositato nella variabile S$ che viene trasferita con l'istruzione PUT, nel file nuovo. È importante notare come questa operazione sia stata resa possibile rispetto alle vecchie versioni di Basic, in quanto è possibile, con il Quick Basic trattare i file in modo binario ed è possibile definire stringhe con lunghezza fino a 32767 caratteri.

Sia nel modo random che binario, se viene tentata la GET ed il record di dati non esiste, non viene letto alcunché e la funzione EOF ritorna il valore -1 (vero).

(Istruzione grafica)

In grafica tale istruzione, insieme alla corrispondente PUT, viene usata principalmente per l'animazione. La sintassi è la seguente

GET [**STEP**] **(x1,y1)-**[**STEP**] **(x2,y2),array**[**(index)**]

Le coordinate x1,y1 e x2,y2 sono, rispettivamente, quelle dei due angoli opposti di un rettangolo sullo schermo grafico; tali coordinate possono essere assolute (senza l'opzione STEP) o relative (con l'opzione STEP); in quest'ultimo caso, se il termine STEP viene posto prima delle x1,y1 esse sono intese come spostamenti relativi all'ultimo punto tracciato; se è presente anche l'opzione STEP nelle coordinate x2,y2, esse sono definite relativamente alle prime; ad esempio le due scritture seguenti, sono equivalenti

GET (100,100)-(150,120),A%
GET (100,100)-STEP (50,20),A%

in quanto, nel secondo caso le coordinate effettive del secondo punto sono determinate dalla somma dei valori specificati con quelli, rispettivi, del primo punto.

L'area di schermo grafico definita in questo modo, viene letta e depositata all'interno dell'array il cui nome è specificato in seguito; se viene specificato il valore di uno o più indici dell'array, i dati sono depositati ad iniziare dall'elemento prescelto. È possibile utilizzare qualsiasi tipo di array purché numerico; la grandezza dell'array utilizzato dipende dal suo tipo, dalla grandezza dell'area definita, dal tipo di modalità grafica adottata e dalla scheda grafica posseduta secondo la seguente formula

(4 + INT((Xdim * Bppps + 7) / 8) * Str * Ydim) / Prec

in cui i parametri utilizzati sono

Xdim dimensione x dell'area definita dal valore dell'espressione x2-x1+1

Ydim dimensione y dell'area definita dal valore dell'espressione y2-y1+1

Bppps valore caratteristico per la modalità e la scheda grafica utilizzate

Str valore caratteristico per la modalità e la scheda grafica utilizzate

Prec valore che dipende dal tipo di array utilizzato

I valori Bppps e Str sono definiti nella seguente tabella

Modalità grafica	Bppps	Str
1	2	1
2	1	1
7	1	4
8	1	4
9	1	2 o 4 (*)

10	1	2
11	1	1
12	1	4
13	8	1

(*) 2 se scheda EGA con memoria = 64 K
4 se scheda EGA con memoria > 64 K

mentre i valori di Prec sono i seguenti

Prec	Tipo array
2	Intero
4	Lungo
4	Singola precisione
8	Doppia precisione

Oltre all'animazione, l'istruzione GET in grafica (insieme alla PUT) è utile per poter conservare e riprendere, tramite disco, informazioni grafiche relative al programma utilizzato; per conservare su disco, per esempio, un'immagine di un triangolo interno ad un cerchio, può essere utilizzato il programma seguente

```
DIM P#(1 TO 103)
SCREEN 1
LINE (70, 75)-(130, 125),2,BF
CIRCLE (100, 100), 30
PSET (100, 100)
LINE (70, 100)-(130, 100)
LINE –STEP(-30, 25)
LINE –STEP(-30, -25)
PAINT STEP(10, 15), 1, 3
PAINT STEP(20, 0), 2, 3
PAINT STEP(20, 0), 1, 3
PAINT (100, 90), 3, 3
GET (70, 75)-(130, 125), P#(1)
DEF SEG=VARSEG(P#(1))
BSAVE "IMAGE.SCR", VARPTR(P#(1)), 103*8
DEF SEG
```

L'espressione 103*8 usata nell'istruzione BSAVE, indica che l'array da salvare è composto da 103 elementi e che ognuno di questi occupa 8 bytes, per un totale di 824 bytes di memoria da registrare su file ad iniziare da quello che contiene il primo elemento (VARPTR(P#(1))) nel segmento specifico (VARSEG(P#(1))). Per definire la grandezza dell'array da utilizzare è più comodo procedere per tentativi dato che il Quick Basic emette un errore se tale grandezza non è sufficiente.
Una volta eseguito il precedente programma, l'immagine è memorizzata su disco ed è utilizzabile con un programma come il seguente

```
DIM L#(1 TO 103)
DEF SEG=VARSEG(L#(1))
BLOAD "IMAGE.SCR", VARPTR(L#(1))
SCREEN 1
FOR T=1 TO 5
  PUT (RND * 230, RND * 150), L#(1)
NEXT T
END
```

La procedura vista in precedenza è largamente utilizzata per conservare immagini su disco da utilizzare in un programma successivo (generalmente un gioco).

*** GOSUB**

Con questa istruzione, classica del linguaggio Basic, è possibile richiamare una sottoroutine che, dopo essere stata eseguita, restituisce il comando al programma chiamante tramite l'istruzione RETURN. La sintassi dell'istruzione GOSUB, in Quick Basic, prevede l'utilizzo alternativo di un numero di riga o di una etichetta simbolica come riferimento al punto di inizio della sottoroutine, e cioè

GOSUB {**numerodilinea** | **etichetta**}

Il numero di linea specificato o l'etichetta, deve esistere e deve essere unico in maniera che Quick Basic possa eseguire correttamente il programma che usa l'istruzione suddetta. Generalmente le sottoroutines sono usate quando, in un programma, una parte di esso deve essere eseguibito diverse volte in diversi punti dello stesso; in questo caso, per evitare di appesantire la parte principale del programma, viene scritta una sottoroutine che può essere richiamata con la GOSUB. Nell'esempio successivo si vede come tale operazione sia possibile in Quick Basic

```
' (Main Module)
CLS
INPUT "SCRIVERE UN NUMERO TRA 1 E 15000 ", N1%
X%=N1%
GOSUB CONVERT
INPUT "SCRIVERNE UN ALTRO ", N2%
X%=N2%
GOSUB CONVERT
PRINT
PRINT "LA SOMMA E' ";N1%+N2%
X%=N1%+N2%
GOSUB CONVERT
END
' (Sottoroutine CONVERT)
CONVERT:
PRINT "(IN ESADECIMALE ";HEX$(X%);")"
RETURN
```

L'istruzione RETURN provvede a far tornare il controllo al programma chiamante proprio all'istruzione seguente la GOSUB appena eseguita; in alcuni casi è possibile fare in modo che l'istruzione RETURN faccia ritornare l'esecuzione del programma ad un punto particolare del programma chiamante; ciò è possibile specificando un numero di linea o una etichetta subito dopo la stessa istruzione, ma in questo modo, bisogna porre particolare attenzione allo stack le cui dimensioni devono essere tali da poter garantire le operazioni suddette. La sintassi dell'istruzione RETURN è quindi, la seguente

RETURN [{numerodilinea | etichetta}]

L'istruzione GOSUB può essere annidata; in una sottoroutine può esistere la chiamata ad un'altra, anche a sé stessa e la dimensione dello stack è l'unico fattore che limita questo uso delle istruzioni GOSUB..RETURN. Di seguito è fornito un programma di esempio di chiamata a sottoroutine ricorsiva

```
CLS
DO WHILE 1
    PRINT
    INPUT "VALORE ", A
    A=INT(A)
    SELECT CASE A
        CASE IS > 170
            PRINT "VALORE TROPPO GRANDE"
        CASE IS = 0
            END
        CASE ELSE
            AX=A
            K#=1
            GOSUB FATT
            PRINT AX; "!="; K#
    END SELECT
LOOP
END
FATT:
    IF A>1 THEN
        K#=K#*A
        A=A-1
        GOSUB FATT
    END IF
    RETURN
```

In questo programma viene calcolato il fattoriale di un numero compreso tra 1 e 170 come nel programma di esempio relativo all'istruzione CLEAR, con la differenza che, usando l'istruzione GOSUB, la dimensione dello stack non deve essere variata, in quanto viene usato in maniera minore, e la velocità di esecuzione è maggiore.

Nel caso in cui l'istruzione RETURN venga usata con un numero di linea o etichetta, queste devono riferirsi alla sottoroutine o al programma in cui era presente la relativa GOSUB per evitare problemi con lo stack.

Per passare dei parametri alle sottoroutines richiamate con l'istruzione GOSUB è sufficiente servirsi delle variabili usate dal programma chiamante tenendo presente che queste non sono considerate locali all'interno della sottoroutine. È questo uno dei motivi che rendono preferibile l'uso delle Subs e delle Functions messe a disposizione dell'utente da Quick Basic.

*** GOTO**

L'istruzione GOTO è stata forse la più usata in versioni precedenti del linguaggio Basic per poter modificare il flusso di operazioni svolte in un programma in maniera incondizionata. Infatti tramite tale istruzione, è possibile, senza alcuna condizione, passare ad eseguire una parte di programma qualsiasi a patto che questa si trovi allo stesso livello di quella in cui è contenuta l'istruzione GOTO. Non è possibile saltare quindi all'interno di Subs o Functions dal livello modulo e non è consigliabile (in certi casi non è consentito) tramite questa istruzione, uscire dalle strutture di controllo come FOR..NEXT, DO..LOOP, SELECT CASE..END SELECT, DEF FN..END DEF, WHILE..WEND. L'istruzione prevede un numero di linea o una etichetta simbolica a cui riferirsi per il salto, e la sua sintassi è la seguente

GOTO {**numerodilinea** | **etichetta**}

L'uso di questa istruzione deve comunque essere limitato ai casi in cui non se ne può fare a meno, preferendo negli altri l'uso delle strutture di controllo indicate in precedenza che permettono una stesura di un programma più lineare e facilitano la correzione dello stesso; non è infatti facile seguire lo svolgimento di un programma, alla ricerca di un errore, se in questo sono presenti molte istruzioni GOTO. Ad esempio, notare come, tra le due seguenti versioni dello stesso programma che genera 5 sequenze di numeri da 1 a 100 di lunghezza casuale, la seconda risulti la più elegante e la più facilmente leggibile

```
' (1^ versione G.BAS con istruzione GOTO)
RANDOMIZE TIMER
CLS
FOR P=1 TO 5
    FOR T=1 TO 100
Entra:
             PRINT T;
             IF RND>.8 THEN
                      GOTO Esce
             END IF
    NEXT T
Esce:
T=1
PRINT
NEXT P
END

' (2^ versione G.BAS senza istruzione GOTO)
```

```
RANDOMIZE TIMER
CLS
FOR P=1 TO 5
    FOR T=1 TO 100
            PRINT T;
            IF RND>.8 THEN EXIT FOR
    NEXT T
    PRINT
NEXT P
END
```

Da notare che alcune istruzioni (come EXIT FOR) sono disponibili in Quick Basic proprio per evitare che le strutture di controllo vengano abbandonate tramite l'istruzione GOTO.

*** IF**

È l'istruzione che consente l'esecuzione condizionale di parti di programma. È previsto l'uso di due forme: la monoriga e la forma a blocco. La forma monoriga è quella classica del linguaggio Basic e la sua sintassi è la seguente

IF esprlogica {THEN | GOTO} parte1 [ELSE parte2]

in cui

esprlogica è un'espressione aritmetico-logica in cui il risultato può essere soltanto vero o falso; si possono usare tutti gli operatori aritmetici (+, -, *, /, ^) e quelli relazionali (<=>); un risultato numerico eguale a 0 viene considerato falso, altrimenti vero;

parte1 sono le istruzioni che seguono la parola THEN o GOTO e sono quelle che vengono eseguite nel caso in cui esprlogica sia vera; nel caso in cui venga usata la forma IF..GOTO, parte1 deve essere una etichetta o un numero di linea: se viene usata la forma IF..THEN, parte1 può essere una qualsiasi istruzione del linguaggio o un numero di linea a cui saltare (è ammessa infatti la forma IF..THEN..numerodilinea che sottintende l'istruzione GOTO);

parte2 è quella parte della riga che contiene le istruzioni da eseguire nel caso in cui esprlogica sia falsa; può essere un numero di linea a cui saltare (in questo caso è sottintesa l'istruzione GOTO) o una qualsiasi istruzione del linguaggio; se questa parte manca ed esprlogica è falsa, la riga IF..THEN.. viene semplicemente ignorata e l'esecuzione continua con le istruzioni seguenti.

Sia parte1 che parte2 possono essere costituite da più istruzioni separate dal simbolo apposito (:) ed altre istruzioni IF..THEN..ELSE possono fare parte.

Un esempio utilizzante questa forma è costituito dal seguente programma che controlla che siano presenti le interfacce seriali (COM1,2,3 e 4)

```
CLS
DEF SEG=&H40
FOR T=0 TO 3
```

```
    K = PEEK(T*2) + PEEK(T*2+1)*256
    C$ = "COM" + STR$(T) + " "
    IF K>0 THEN PRINT C$;"PRESENTE" ELSE PRINT C$;"NON INSTALLATA"
NEXT T
DEF SEG
```

La seconda forma prevede una sintassi diversa contempla l'uso delle istruzioni END IF ed ELSEIF

```
IF esprlogica1 THEN
    Blocco1
[ELSEIF esprlogica2]
    [blocco2]
    …
[ELSEIF esprlogican]
    [bloccon]
    …
[ELSE]
    [bloccoe]
END IF
```

in cui

esprlogica1, esprlogica2…esprlogican sono delle espressioni aritmetico-logiche che ritornano un valore che può essere vero o falso; sono del tutto simili alla esprlogica usata nella prima forma.

blocco1 sono delle istruzioni che vengono eseguite solo se la esprlogica1 è vera; in questo caso, dopo l'esecuzione del blocco riprende dopo l'istruzione END IF che chiude tutta la struttura;

blocco2..bloccon sono delle istruzioni che vengono eseguite solo se la esprlogica1 è falsa, se sono false tutte le esprlogiche che precedono quella del blocco interessato che è vera; se per esempio, esprlogica1 ed esprlogica2 sono false ed esprlogica3 è vera, viene eseguito blocco3; tutte le clausole ELSEIF sono opzionali, ma, se presenti, vengono prese in considerazione in sequenza e viene eseguito il blocco corrispondente a quello che, per prima, viene valutata vera;

bloccoe è quella parte della struttura che viene eseguita solo se tutte le possibilità precedenti non sono state sfruttate; se la clausola ELSE non viene usata, l'esecuzione del programma riprende comunque dall'istruzione END IF.

Questa forma della struttura IF è molto più flessibile e potente della precedente; è possibile annidare tali strutture, che, con le altre disponibili in Quick Basic, consentono di realizzare programmi efficienti e chiari. Ecco di seguito un esempio di uso della forma a blocchi della IF

```
CLS
DEF SEG=&H40
DO WHILE INKEY$=""
    K=PEEK(&H97)
```

```
        IF K AND 4 THEN
            PRINT "LED DEL CAPS LOCK ACCESO"
        ELSE
            PRINT "LED DEL CAPS LOCK SPENTO"
        END IF
    LOOP
```

In questo caso, viene controllato che il Led presente sulla tastiera riguardante il tasto Caps Lock, sia acceso o spento; notare che l'espressione K AND 4 ritorna un valore che può essere 0 o 4; il primo viene ritenuto falso, l'altro vero.

*** IMP**

È un operatore logico poco usato che rispetta la seguente tabella della verità

1^ Operando	2^ Operando	Risultato
0 (FALSE)	0 (FALSE)	**1 (TRUE)**
0 (FALSE)	**1 (TRUE)**	**1 (TRUE)**
1 (TRUE)	0 (FALSE)	0 (FALSE)
1 (TRUE)	**1 (TRUE)**	**1 (TRUE)**

L'operatore IMP può essere usato in qualsiasi espressione logico-aritmetica ma possiede il più basso livello di priorità tra gli operatori, come del resto, per gli altri operatori logici usabili in Quick Basic.

Ecco, ad esempio, come è possibile negare un valore usando tale operatore

```
CLS
FOR T=1 TO 20
    PRINT T, 1+(T IMP 0)
NEXT T
```

Notare che l'operatore IMP agisce bit per bit sui due operandi.

*** INPUT**

Tramite questa istruzione è possibile immettere, servendosi della tastiera, dati sia numerici che di tipo stringa, e depositarli all'interno delle variabili del tipo opportuno. La sintassi dell'istruzione è la seguente

INPUT [**;**] [**"stringaprompt"**{**;**|**,**}] **elenco variabili**

in cui

; questo simbolo posto, opzionalmente, subito dopo l'istruzione INPUT, fa in modo che, dopo l'input dei dati, il cursore non vada alla prossima linea;

"stringaprompt" è una stringa costante che, se indicata, viene visualizzata alla posizione del cursore prima dell'immissione dei dati richiesti; se, di seguito, viene specificato il simbolo

(;), allora viene presentato di seguito un punto interrogativo; se viene specificata una virgola (,), il punto interrogativo non viene evidenziato;

elencovariabili è l'elenco di variabili in cui verranno depositati i dati immessi; se nell'elenco compare più di una variabile, queste devono essere separate da una virgola.

Fare attenzione al fatto che i dati immessi da tastiera devono essere dello stesso tipo delle variabili che li devono ricevere; nel caso in cui i dati in ingresso fossero più di uno, essi vanno separati tramite una virgola che quindi, normalmente, non può fare parte di un dato alfanumerico; tuttavia, è possibile, ma non necessario, includere quest'ultimo tipo di dati tra virgolette in modo da includere anche spazi e virgole all'interno di essi, come nel seguente esempio

```
CLS
INPUT "Indirizzo : ", I$
PRINT I$
END
```

la cui esecuzione darebbe i seguenti risultati

Indirizzo : "Corso Italia, 54"
Corso Italia, 54

(infatti, l'immissione dell'indirizzo compreso tra virgolette, permette di accettare anche la virgola senza alcun errore).

Nel caso in cui viene immesso un dato in meno rispetto al previsto, allora viene evidenziata la frase di avvertimento "Redo from start" (*Ricomincia dall'inizio*, nella versione 4.5) che avverte, dunque, che l'operazione di input deve essere ripetuta. Questo potrebbe, in un programma applicativo, risultare scomodo, tanto che si preferisce, quasi sempre, provvedere all'input dei dati con delle routines personalizzate.

Non è possibile immettere più di 255 caratteri in un solo input e durante l'immissione dei dati, sono funzionanti i tasti di editing comunemente usati; ecco di seguito una lista delle funzioni dei più importanti

Home	porta il cursore all'inizio dell'input
End	porta il cursore alla fine dell'input
Ins	attiva/disattiva l'inserimento
Esc	annulla l'input già fornito

*** INPUT #**

Con questa istruzione, possono essere letti dei dati dai file sequenziali o da periferiche precedentemente aperti tramite l'istruzione OPEN. La sintassi dell'istruzione

INPUT #numcanale, elenco variabili

prevede l’immissione del numero di canale aperto (numero riportato nell’istruzione OPEN corrispondente), e l’elenco delle variabili in cui devono essere depositati i dati letti. Questi ultimi devono arrivare dal dispositivo di lettura, in maniera che i numeri siano divisi tramite uno spazio, una virgola o un carattere di ritorno a capo e le stringhe da una virgola o da un carattere di ritorno a capo; anche il carattere di fine file viene considerato come terminatore per un dato letto da tale istruzione. Se si tenta di leggere dati dopo che il carattere di fine file è stato incontrato, viene emesso un errore di tipo “Input past end” (Input oltre la fine nella versione 4.5). Notare che il carattere #, nella sintassi dell’istruzione, è obbligatorio.

*** INTEGER**

Vedere AS, COMMON, DECLARE, DEF FN, DIM, FUNCTION, SHARED, STATIC, SUB, TYPE

*** IOCTL**

Questa istruzione è utilizzata, insieme alla funzione IOCTL$, per gestire i device drivers. La sintassi di questa istruzione è la seguente

IOCTL [**#**]**canale, stringa**

Il primo parametro indica il numero di canale con cui è stata eseguita l’istruzione OPEN relativa all’apertura del flusso di comunicazione con il device driver, mentre, il secondo, è la stringa di comando inviata al device driver. Dato che questa istruzione è dipendente in maniera elevata dal device driver che è stato installato e per cui si usa, viene fornito solo un esempio generico di trasmissione di una stringa di comando al device driver

```
OPEN … AS #1          ‘ Apertura del canale con il device driver
IOCTL #1, “…”         ‘ Stringa di comando inviata al d. d.
CLOSE #1
END
```

Tenere presente che nessun device driver delle periferiche standard del DOS gestiscono le stringhe di comando IOCTL.

*** IS**

Vedere SELECT CASE

*** KEY**

L’istruzione KEY, usata per definire il contenuto dei tasti funzione, è utilizzabile in diverse maniere le cui sintassi sono elencate di seguito

KEY {**ON** | **OFF**}

KEY LIST

KEY numtasto, stringa

La prima sintassi è quella relativa all’abilitazione della visualizzazione del contenuto dei 10 tasti funzione durante l’esecuzione di un programma; l’opzione ON abilita tale funzione mentre

l’opzione OFF la disabilita; normalmente quest’ultima opzione è intesa per default. Durante un input, tramite i tasti Ctrl T, è possibile visualizzare i tasti funzione o no.

Ad ogni tasto funzione, da F1 a F10 (o F12 per le tastiere estese), può essere fatta corrispondere ad una stringa la cui lunghezza massima non può superare i 15 caratteri; comunque, tramite l’istruzione KEY ON, vengono visualizzati solo i primi 6 caratteri di ogni tasto funzione compreso tra F1 e F10; per controllare il contenuto completo di tutti i tasti funzione (inclusi F11 e F12), è possibile utilizzare la seconda sintassi dell’istruzione, KEY LIST, che elenca a video tutte le informazioni richieste.

Per associare una stringa ad un tasto funzione, viene usata la terza sintassi dell’istruzione KEY; numtasto è un numero corrispondente ad alcuni tasti che è possibile definire, elencati nella seguente tabella

1-10	Tasti funzione da F1 a F10
11	SU
12	SINISTRA
13	DESTRA
14	GIU’
15-25	Tasti definibili dall’utente
30-31	Tasti funzione F11 e F12

mentre il parametro stringa seguente la virgola di separazione, è la sequenza di caratteri associata al tasto prescelto. Ad esempio, se si volesse fare corrispondere la parola Fine al tasto funzione numero 10, bisognerebbe fornire la seguente riga di istruzione

KEY 10, “Fine”

Notare che, premendo il tasto funzione 10, avvengono le stesse cose che avverrebbero se si fosse introdotta la parola Fine da tastiera; se si volesse simulare anche il tasto Return, la precedente riga dovrebbe essere modificata così

KEY 10, “Fine”+CHR$(13)

Se la tastiera usata è del tipo esteso, tramite i codici 30 e 31, è possibile definire i tasti funzione F11 e F12. I codici che sono compresi tra 15 e 25 sono a disposizione dell’utente che li può definire a proprio piacimento, servendosi, per sfruttarli, dell’istruzione ON KEY(n); a tale scopo viene usata una particolare versione della terza sintassi del comando KEY, che prevede che la stringa da assegnare sia costituita da due caratteri; il significato del valore ASCII del primo dei due caratteri è specificato nella tabella seguente

0	nessun tasto di Shift della tastiera
1	Shift sinistro
2	Shift destro
4	Ctrl
8	Alt
32	Num Lock
64	Caps Lock

128 Tasti della tastiera avanzata

Sommando questi valori, sono rilevate le varie combinazioni di tasti. Per rilevare l'uso di uno qualsiasi dei due tasti Shift, ad esempio, è possibile usare il codice 3.
Il secondo valore è invece relativo al codice di tastiera del tasto che si vuole rilevare, secondo la tabella seguente

Tasto	Codice	Tasto	Codice	Tasto	Codice
Esc	1	Ctrl	29	Spazio	57
! o 1	2	A	30	Caps Lock	58
@ o 2	3	S	31	F1	59
# o 3	4	D	32	F2	60
$ o 4	5	F	33	F3	61
% o 5	6	G	34	F4	62
^ o 6	7	H	35	F5	63
& o 7	8	J	36	F6	64
* o 8	9	K	37	F7	65
(o 9	10	L	38	F8	66
) o 0	11	: o ;	39	F9	67
_ o -	12	" o '	40	F10	68
+ o =	13	~ o '	41	Num Lock	69
Bckspc	14	Shift sx	42	Scroll Lock	70
Tab	15	| o \	43	Home o 7	71
Q	16	Z	44	Su o 8	72
W	17	X	45	Pg Up o 9	73
E	18	C	46	-	74
R	19	V	47	Sinistra o 4	75
T	20	B	48	5	76
Y	21	N	49	Destra o 6	77
U	22	M	50	+	78
I	23	< o ,	51	End o 1	79
O	24	> o .	52	Giù o 2	80
P	25	? o /	53	Pg Dn o 3	81
{ o [	26	Shift dx	54	Ins o 0	82
} o]	27	Prtscr o *	55	Del o .	83
Return	28	Alt	56		

Ad esempio, per intercettare il in qualsiasi momento, il tasto Escape, è disponibile la seguente routine d'esempio

```
CLS
KEY 15, CHR$(0)+CHR$(1)          ' Codice tasto = 1
ON KEY(15) GOSUB ESCAPE
KEY(15) ON
DO WHILE A$<>CHR$(13)
    A$=INKEY$
    PRINT A$;
LOOP
```

```
END
ESCAPE:
    PRINT "È STATO PREMUTO IL TASTO ESCAPE"
RETURN
```

Tenere presente che, durante un input, l'intercettazione del tasto viene sospesa e quindi, se si fosse usata l'istruzione INPUT al posto della INKEY$, non si sarebbe notato il funzionamento del tasto Escape.

I codici che vanno da 11 a 14 sono usabili sempre, tramite l'istruzione ON KEY(n), a cui si rimanda per ulteriori chiarimenti, e servono a gestire i tasti di controllo del cursore nelle 4 direzioni.

*** KEY(n)**

Vedere ON KEY(n), KEY

*** KILL**

Questa istruzione si comporta esattamente come il comando del DOS Erase o il comando Del; viene usata in Quick Basic per eliminare da una directory di un disco specificato, il file o il gruppo di file indicati. Ecco la sintassi del comando KILL

KILL esprstringa

in cui esprstringa è un'espressione di tipo alfanumerico che contiene il nome del file da eliminare, eventualmente completo di drive e directory dove lo stesso si trova; è possibile usare i caratteri jolly (* e ?) usati dal DOS per eliminare un gruppo di file. Se il file o i file specificati non esistono, viene generato un errore specifico; non è consentito usare questa istruzione con file che sono ancora aperti. La seguente funzione di esempio, mostra un uso dell'istruzione KILL; essa serve a controllare l'esistenza di un determinato file specificato come argomento; la stessa ritorna un valore logico (False o True) a seconda se il file è presente o no

```
FUNCTION FILEEXIST%(PATH$) STATIC
    DEFINT E
    OPEN "R",#120,PATH$
    IF LOF( 120)=0 THEN
            EX=0
    ELSE
            EX=1
    END IF
    CLOSE #120
    KILL PATH$
    FILEEXIST%=EX
END FUNCTION
```

*** LET**

La sintassi di questa istruzione, unica in Quick Basic, prevede che la parola chiave LET sia facoltativa; infatti

[**LET**] **elemvar = espressione**

è la sintassi dell'istruzione LET in cui la stessa parola chiave può essere omessa dato che il segno di uguale, in questo caso, viene identificato come simbolo di assegnazione ed in quanto tale è sufficiente per una corretta compilazione della linea. L'uso della parola chiave LET viene mantenuto solo per una questione di portabilità del codice sorgente di quei programmi scritti nelle versioni di Basic che prevedevano tale istruzione in modo obbligatorio, versioni di Basic quasi del tutto fuori uso.
È importante che la variabile indicata a sinistra del segno si assegnazione sia dello stesso tipo del risultato fornito dall'espressione posta alla destra dello stesso segno. La variabile può essere di tipo semplice, array o di tipo definito dall'utente ma, in quest'ultimo caso, deve essere specificato un solo elemento definito nel tipo utente.

Ad esempio, per le variabili semplici, i seguenti esempi sono validi ed equivalenti

LET A#=1D3
oppure
A#=1D3

LET A$="Quick Basic"
oppure
A$="Quick Basic"

lo stesso per gli elementi di array

LET A%(5)=38
oppure
A%(5)=38

o per gli elementi di una variabile di tipo utente

```
TYPE RECANA
    COGNOME AS STRING * 20
    NOME AS STRING * 20
    ETA AS INTEGER
END TYPE
DIM PERSONA AS RECANA
```

LET PERSONA.COGNOME = "Rossi"
oppure
PERSONA.COGNOME = "Rossi"

È buona norma non usare tale istruzione per non appesantire inutilmente il testo sorgente.

*** LINE**
La sintassi di questo comando grafico è il seguente

LINE [[**STEP**] **(x1,y1)**]-[**STEP**] **(x2,y2)** [,[**color**][,[**B**[**F**]][**,style**]]]

in cui x1,y1 e x2,y2 sono due coppie di coordinate di punti dello schermo grafico che specificano gli estremi del segmento da tracciare. Queste coordinate sono, normalmente, indicate in maniera assoluta mentre, se sono precedute dalla parola STEP, sono intese relative alle coordinate attuali del cursore grafico che dipendono dall'ultima istruzione grafica eseguita che le ha modificato. La prima coppia di coordinate è opzionale e, se manca, si intende eguale a quella della posizione attuale del cursore grafico; con questo uso si possono tracciare poligoni e figure composte da molti segmenti molto facilmente.

Il parametro color, se presente, rappresenta il colore con cui il segmento verrà tracciato sullo schermo, colore che dipende dal tipo di modalità grafica, scheda video e monitor usati (vedere COLOR e SCREEN). Per default, il colore usato è quello di foreground.

L'opzione B, quando specificata, permette di tracciare automaticamente il rettangolo la cui diagonale sia stata specificata tramite le due coppie di coordinate di punti; in questo caso viene tracciato solo il rettangolo e non la diagonale. Inoltre, se invece viene specificato il parametro BF, l'interno del rettangolo viene colorato automaticamente con il colore specificato in precedenza.

Infine, tramite l'opzione style, è possibile utilizzare un tipo personalizzato di linea continua, tratteggiata, a punti o comunque si preferisca. Per realizzare ciò è sufficiente specificare in questo punto un numero intero formato da 16 bits la cui maschera rappresenta il tipo di linea usata; ad esempio, per usare un tipo di linea tratto-punto, il valore da specificare deve essere il seguente

```
*** *  *** * ***
1110101110101110 = &HEBAE
```

che usato nella linea

```
LINE (10,10)-(200,100),,B,&HEBAE
```

permette di visualizzare il rettangolo tracciato con il tratto punto-linea, mentre lo stesso rettangolo, con il valore &H5555 per lo style, sarebbe stato tracciato con una sequenza di punti dato che il valore indicato è equivalente alla seguente maschera di bits

```
0101010101010101 = &H5555
 * * * *  * * * *
```

Il prossimo esempio, invece, mostra come tracciare un triangolo i cui lati siano di 3 colori diversi

```
SCREEN 1
LINE (100,150)-(200,100),1
LINE -(150,150),2
LINE -(100,150),3
A$ = INPUT$(1)
```

l'ultima linea è stata aggiunta per permettere di vedere il triangolo fino a che un tasto non viene premuto.

*** LINE INPUT**

Questa istruzione, molto simile alla INPUT, ha la seguente sintassi

LINE INPUT [**;**] [**"stringacost"**{**;** | **,**}]**varstringa**

A differenza della INPUT, la LINE INPUT permette di immettere anche i caratteri come le virgolette e la virgola senza influenzare in alcun modo l'input. L'unico modo per interrompere tale input è il tasto Return. La lunghezza massima della linea di dati che è possibile immettere con una unica LINE INPUT è 255. Tali dati sono depositati all'interno della 'varstringa', variabile di tipo stringa, elemento obbligatorio nella sintassi dell'istruzione.
Se specificata, è possibile usare una stringa costante come primo argomento, per la visualizzazione di un messaggio prima dell'input; se tale messaggio è variabile, si deve usare l'istruzione PRINT per visualizzare e poi la LINE INPUT per introdurre i dati. La costante suddetta può anche essere vuota.
L'uso della virgola o del punto e virgola tra la stringa costante e la variabile, è indifferente.
Se viene usato il punto e virgola subito dopo l'istruzione, non viene spostato il cursore dalla riga di inserimento dopo la conclusione dell'input. Per usare tale opzione non è necessario usare la stringa costante.

Il seguente esempio mostra l'uso di tale istruzione; il programma visualizza in maniera inversa, la frase immessa da tastiera

```
DEFINT A-Z
CLS
LINE INPUT "FRASE : ", A$
IF LEN(A$) >0 THEN
        FOR X=LEN(A$) TO 1 STEP -1
                PRINT MID$(A$, X, 1);
        NEXT X
END IF
END
```

*** LINE INPUT #**

Similmente alla precedente istruzione, questa permette di immettere una variabile stringa, dei caratteri, fino a 255, senza considerare altri delimitatori tranne il carriage-return; questo formato però, accetta l'input da file secondo la seguente sintassi

LINE INPUT #numfile, varstringa

in cui 'numfile' è il numero del file a cui si fa riferimento e 'varstringa' il nome della variabile alfanumerica in cui devono essere depositati i dati in input.
Questa istruzione è, per esempio, usata per leggere il contenuto di file sequenziali di cui non si conosce la struttura; questa funzione è appunto svolta dal seguente programma di esempio

```
DEFINT A-Z
CLS
INPUT "NOME FILE : ",FR$
```

```
OPEN FR$ FOR INPUT AS #1
PRINT
DO WHILE NOT EOF(1)
        LINE INPUT #1, R$
        PRINT R$
LOOP
CLOSE
END
```

*** LOCAL**

Questa istruzione, seppure riservata nelle due versioni di QB (4.0 e 4.5), non è usata in nessuna delle due ed è pronta per sviluppi futuri.

*** LOCATE**

Questa istruzione è usata, in modo testo, per modificare la posizione, attivare e modificare la forma del cursore. La sintassi di LOCATE è la seguente

LOCATE [**riga**][,[**colonna**][,[**viscurs**][,[**inizio, fine**]]]]

in cui tutti i parametri sono opzionali; ognuno di questi, se mancante, viene automaticamente definito con il valore corrente che viene sempre conservato. Per quanto riguarda riga e colonna esse sono limitate al formato dello schermo (generalmente 25 righe di 80 caratteri) ma queste possono variare con l'uso di diverse schede video; la 25^ riga viene utilizzata solo se viene specificata in precedenza l'istruzione KEY OFF che annulla la visualizzazione del contenuto dei tasti cursore.

viscurs è un parametro che, se posto a zero, rende il cursore invisibile (ne disattiva la visualizzazione) mentre, posto ad un valore diverso da zero, lo rende visibile.

La forma del cursore stesso dipende dagli ultimi due valori di cui il primo indica il numero di linea dalla quale deve iniziare ad essere visualizzato lo stesso e la seconda, la linea alla quale deve terminare la visualizzazione. Scegliendo valori opportuni, il cursore può essere eliminato dallo schermo senza usare il parametro viscur.

Ecco di seguito, un esempio d'uso dell'istruzione LOCATE per la visualizzazione, anche se approssimativa, della funzione trigonometrica seno in modo testo

```
CONST PIGRECO=3.14159
CLS
X=1
COLOR 15
FOR T=0 TO 5*PIGRECO STEP .4
        LOCATE SIN(T)*10+13, X
        PRINT "*";
        X=X+2
NEXT T
COLOR 7
END
```

*** LOCK**

A partire dalla versione 3.1 di MS-DOS, tramite il comando Share, è possibile che i programmi applicativi gestiscano file di dati condivisi da altri programmi in esecuzione su terminali collegati tramite rete. Per evitare il conflitto tra i due programmi, se questi accedono contemporaneamente agli stessi archivi di dati, è possibile bloccare parti, di dimensioni variabili, di file. Questa operazione è realizzabile tramite l'uso dell'istruzione LOCK la cui sintassi è la seguente

LOCK [**#**]**numfile,** [{**record** | [**inizio**] **TO file**}]

in cui è obbligatorio il numero del file a cui si fa riferimento, numero che è indicato nell'istruzione OPEN corrispondente. Se non viene specificato altro nella linea di comando, tutto il file viene bloccato in maniera che altri non possono modificarlo; infatti il file viene bloccato in scrittura ma non in lettura così da consentire ad altri programmi di controllarne il contenuto; dopo che il file viene modificato, lo stesso viene nuovamente reso disponibile tramite l'istruzione complementare UNLOCK. Possono essere invece specificati altri due parametri, inizio e fine, di cui il primo è opzionale, e che, a seconda del tipo di istruzione OPEN usata per aprire il file considerato, sono interpretati in maniera diversa. Se il file è di tipo RANDOM, i valori di inizio e fine sono intesi come numero di record dal quale inizia ed al quale termina la protezione dalla scrittura; ad esempio, specificando la linea

LOCK #1, 10 TO 50

ed il file numero 1 è stato aperto in modo RANDOM, vengono bloccati tutti i record compresi tra il numero 10 ed il numero 50; è possibile non indicare il record iniziale ed in questo caso viene inteso il numero 1. Se infine viene indicato un solo valore, si intende il record che si vuole bloccare.

Nel caso in cui il file venga aperto in modo BINARY, i valori di cui sopra si riferiscono alla posizione dei bytes all'interno del file considerando il byte iniziale come il numero 1; se si volesse, ad esempio, bloccare un file aperto in tale modo, i caratteri che vanno dal numero 100 al numero 200, è necessario usare la seguente linea

LOCK #1, 100 TO 200

Se il file aperto in modo sequenziale (INPUT, OUTPUT, APPEND), i valori specificati nell'istruzione LOCK sono ignorati ed il file viene bloccato interamente.

È molto importante, per conoscere le modalità di protezione adottate per un determinato file, controllare anche la modalità di apertura effettuata tramite l'istruzione OPEN a cui si rimanda per ulteriori chiarimenti.

Dopo aver bloccato una parte o tutto un file, prima che lo stesso venga chiuso tramite l'istruzione CLOSE, è necessario usare l'istruzione UNLOCK che permetta di sbloccarlo. È consigliabile, per evitare situazioni impreviste, adottare con l'istruzione UNLOCK gli stessi valori adottati in precedenza con l'istruzione LOCK. La sintassi dell'istruzione **UNLOCK** infatti, è identica a quella della LOCK e cioè

UNLOCK [**#**]**numfile,** [{**record** | [**inizio**] **TO file**}]

Nel caso in cui le due istruzioni (LOCK ed UNLOCK) vengano usate con una versione di DOS non adatta, viene presentato un messaggio di errore; un altro messaggio di errore viene manifestato se si tenta di accedere in scrittura, o comunque in un modo vietato, ad un file già aperto da un programma. Tramite la gestione di tale errore (Access denied), con l'istruzione ON ERROR GOTO, è possibile permettere un accesso multiplo e controllato di diversi processi agli stessi archivi.

*** LONG**

Vedere AS, COMMON, DECLARE, DEF FN, DIM, FUNCTION, SHARED, STATIC, SUB, TYPE

*** LOOP**

Vedere DO

*** LPRINT**

Vedere PRINT

*** LPRINT #**

Vedere PRINT #

*** LPRINT USING**

Vedere PRINT USING

*** LSET**

L'istruzione LSET è usata per predisporre all'interno di una variabile di tipo stringa il risultato di una espressione alfanumerica allineando quest'ultimo a sinistra. Anche se questa operazione può essere effettuata con una qualsiasi variabile di tipo alfanumerico, l'istruzione LSET, e la sua complementare RSET, vengono usate per preparare le variabili definite come campi costituenti un record di file random. Anzi, per questa funzione, le istruzioni LSET e RSET sono le sole disponibili dati che la LET (o semplicemente il segno '=' di assegnazione) non risulta adatta allo scopo. La sintassi delle due istruzioni sono le seguenti

LSET varstringa=esprstringa

RSET varstringa=esprstringa

Mentre l'istruzione LSET allinea i dati a sinistra, la RSET li allinea a destra, anche se ambedue troncano i dati a sinistra se la lunghezza degli stessi è maggiore di quella della variabile di arrivo. Ad esempio, se si volesse predisporre allineandoli a sinistra, la parola CIAO in una variabile lunga 20 caratteri, si dovrebbero eseguire le seguenti linee di istruzioni

```
A$=SPACE$(20)
LSET A$="CIAO"
```

mentre, per allineare a destra un valore numerico contenuto nella variabile X, all'interno di una variabile di tipo stringa lunga 10 caratteri, sono necessarie le linee

```
A$=SPACE$(10)
RSET A$=STR$(X)
```

Se, nel primo caso, la lunghezza della variabile di arrivo fosse di 2 caratteri, sia l'istruzione LSET che la RSET vi copierebbero solo i primi due caratteri (CI).

Per potere trasferire nel buffer contenente il record di un file random i dati che si intendo scrivere sullo stesso, è necessario operare una istruzione LSET (o RSET) per ogni campo elencato nell'istruzione FIELD.

Ecco di seguito, le linee di programma necessarie per potere scrivere su file random, nei primi 12 records, il nome e la durata di ogni mese dell'anno

```
CLS
OPEN "R",#1,"MESI",11
FIELD #1,9 AS NOME$, 2 AS GIORNI$
FOR M=1 TO 12
      READ M$,G
      LSET NOME$=M$
      LSET GIORNI$=MKI$(G)
      PUT #1,M
NEXT M
CLOSE
END
DATA Gennaio,31,Febbraio,28,Marzo,31,Aprile,30
DATA Maggio,31,Giugno,30,Luglio,31,Agosto,31
DATA Settembre,30,Ottobre,31,Novembre,30,Dicembre,31
```

*** MID$**

Questa istruzione, da non confondere con l'omonima funzione, serve a modificare una stringa in una sua parte senza doverla creare ex-novo. La sua sintassi è la stessa della funzione omonima, ma invece di estrarre dall'argomento stringa una sua parte, provvede ad assegnare alla stessa un'altra stringa. Ad esempio, per cambiare il contenuto della seguente stringa

```
A$="Essere o non avere, questo e' il problema"
```

in quest'altra

"Essere o non essere questo e' il problema"

basta usare la seguente linea di programma

```
MID$(A$, 14)="essere"
```

Da notare che la parte di stringa sostituita deve avere la stessa lunghezza di quella che sostituisce; non è infatti possibile cambiare il numero dei caratteri all'interno della stringa, se la parte da sostituire non è nella parte terminale della stessa.

*** MKDIR**

Questa istruzione agisce come l'omonimo comando del DOS che crea una nuova directory nel disco specificato. A differenza del comando del DOS, l'istruzione MKDIR del Quick Basic non è usabile in alcuna forma abbreviata. La sua sintassi è la seguente

MKDIR stringanuovadirectory

in cui il parametro con cui si specifica la nuova directory che deve essere creata, non può essere più lungo di 128 caratteri. Se viene specificato un percorso alla fine del quale deve essere creata la nuova directory e tale percorso è errato, viene emesso un messaggio di avvertimento.
Ad esempio, per creare il seguente albero delle directory all'interno del disco contenuto nel drive A

devono essere eseguite le seguenti linee di programma

```
MKDIR "A:\QB45"
MKDIR "A:\GIOCHI"
MKDIR "A:\QB45"
MKDIR "A:\QB45\SORG"
MKDIR "A:\QB45\LIB"
```

*** MOD**

L'operatore aritmetico MOD restituisce il valore del resto della divisione tra due valori interi. Nelle espressioni ha una priorità di valutazione molto bassa, ed è, per lo più, usato in congiunzione con l'operatore di divisione intera (****) che fornisce il quoto di una divisione tra interi.
Nell'esempio seguente, infatti, si può vedere come è possibile ottenere i quoti ed i resti delle divisioni dei numeri da 0 a 11 per 3

```
CLS
FOR X=0 TO 11
    PRINT X \ 3, X MOD 3
NEXT X
END
```

*** NAME**

È l’istruzione di Quick Basic equivalente al comando RENAME del DOS; infatti è usata per cambiare il nome di un file già contenuto all’interno di un disco, in uno che non sia già presente. La sintassi di NAME è la seguente

NAME oldfile AS newfile

in cui oldfile è una stringa contenente il vecchio nome del file che si vuole cambiare, completo di drive e di percorso e newfile è il nome nuovo del file completo di drive e percorso.

Se, ad esempio, si volesse cambiare il nome del file PROVA contenuto nella directory di lavoro del disco C, in ESEMPIO, si dovrebbe eseguire la seguente linea

NAME “C:PROVA” AS “C:ESEMPIO”

Esistono tuttavia delle differenze tra il comando NAME di Quick Basic ed il comando RENAME del DOS; infatti, se il drive su cui si trova il file da rinominare, è indicato nella linea di comando REN, ciò è fatto solo nel primo parametro mentre deve essere fatto in tutti e due i parametri del comando NAME. Inoltre, differenza molto più importante, il comando NAME provvede automaticamente a **spostare** il file indicato da una directory ad un’altra, cambiandogli contemporaneamente il nome; esegue quindi un comando COPY prima di cambiare il nome al file di destinazione e un DEL del vecchio file al fine di eliminarlo; questo modo di funzionamento ricorda molto il comando mv di UNIX e XENIX ed il programma seguente, fornito come esempio, illustra una piccola versione di tale utility che usa la linea di comando del DOS tramite la funzione COMMAND$

```
‘ (Main Module)
PRINT “MOVE Vrs. 2.0. – A. Giuliana 1990”
S1=INSTR(COMMAND$, “ “)
S2$=””
IF S1<>0 THEN
    S1$=LEFT$(COMMAND$, S1-1)
    S2$=MID$(COMMAND$, S1+1)
END IF
IF S1=0 OR S2$=”” THEN
    PRINT “Illegal use of MOVE”
    PRINT “Usage : MOVE pathfrom pathto”
    PRINT
    END
END IF
ON ERRORE GOTO ERTRAP
NAME S1$ AS S2$
ON ERROR GOTO 0
PRINT
END
ERTRAP:
    PRINT “Error during MOVE “; S1$; “ as “; S2$
    PRINT
    END
```

Se si volesse compilare questo programma, si dovrebbero eseguire i due semplici comandi elencati in seguito, a livello DOS

```
BC /E MOVE;
LINK MOVE;
```

la prima permette di generare il file MOVE.OBJ dal file MOVE.BAS tenendo conto della gestione degli errori e la seconda permette di creare il file MOVE.EXE, eseguibile sotto DOS.
Se si volesse eseguire il file MOVE.EXE così generato, si renderebbe necessaria la presenza del file BRUN40.EXE (o BRUN45.EXE nella versione 4.5); per fare in modo di rendere autonomo il file MOVE.EXE, MOVE.BAS deve essere compilato con l'opzione /O cioè

```
BC /E /O MOVE;
```

così il comando LINK utilizzerà la libreria BRUN40.LIB (la BRUN45.LIB per la versione 4.5) al posto della BCOM40.LIB (della BCOM45.LIB per la versione 4.5) ed il file MOVE.EXE sarà così autonomo.

*** NEXT**

Vedere FOR

*** NOT**

È l'operatore logico della negazione. È usato per negare un valore bit per bit. L'operatore NOT agisce su un operando secondo la seguente tabella

Operando	Risultato
0 (FALSE)	**1 (TRUE)**
1 (TRUE)	0 (FALSE)

Il valore -1 che, con 16 bits viene rappresentato tramite una sequenza di 1, viene trasformato in 0 da questo operatore e viceversa. È quindi utile per negare un risultato logico di un'espressione come nel seguente esempio

```
CONST FALSE=0, TRUE=NOT FALSE
```

in cui la costante FALSE assume il valore 0 e la costante TRUE il valore -1. Nel caso in cui si usasse con una funzione che restituisce un valore logico, come nel caso della EOF(nf), può essere usato come nel seguente programma di prova

```
CLS
OPEN "I",#1,"PROVA"
DO WHILE NOT EOF(1)
        INPUT #1,A$
        PRINT A$;
LOOP
CLOSE #1
```

END

in cui, dopo avere aperto un file sequenziale, lo stesso viene letto solo finché la funzione EOF(n) non ritorna un valore logico vero.

*** ON ERROR**

Insieme all'istruzione RESUME, alle variabili di sistema ERR ed ERL, l'istruzione ON ERROR assicura una gestione avanzata degli errori che si possono verificare durante l'esecuzione di un programma. La sintassi dell'istruzione

ON ERROR GOTO {**numero riga** | **etichetta**}

richiede che sia specificato un numero di riga o un'etichetta da cui inizia una routine di gestione degli errori. Questa routine si deve trovare a livello modulo e deve essere indipendente dal codice contenuto nello stesso. Per questa ragione, normalmente essa è scritta subito dopo la fine del modulo, dopo l'istruzione END che, in questo caso, si rende necessaria per evitare che la routine di gestione degli errori venga attivata anche senza un errore. Se si specifica 0 come numero di riga nell'istruzione ON ERROR GOTO, la gestione degli errori viene disattivata e, se si verifica un errore, questo viene trattato in maniera normale.
Se una routine di gestione degli errori è attiva e si verifica un errore trattabile, un codice numerico unico per ogni errore, viene ritornato dalla funzione ERR; l'esame di tale funzione da parte della routine permette di effettuare le operazioni necessarie alla sua correzione. Inoltre, dalla funzione ERL, viene ritornato il numero di linea in cui si è verificato l'errore; se la linea non ha numero, ERL ritorna il valore del numero dell'ultima linea eseguita che ne possiede uno e, se nessuna linea precedente ha numero, ERL ritorna 0.

Se l'istruzione ON ERROR GOTO 0 viene specificata, l'errore viene evidenziato con uno dei messaggi d'errore propri del Quick Basic; se un errore si verifica all'interno della routine di gestione, questo non può essere trattato dalla stessa e viene subito evidenziato. Se l'errore si verifica in una Sub o una Function, viene richiamata la routine di gestione degli errori che era stata attivata in precedenza a livello di modulo, anche se la Sub o la Function appartiene ad un altro modulo richiamato in catena.
All'interno delle Subs e delle Functions è possibile attivare le routines di gestione degli errori ma queste si devono trovare comunque al livello di modulo.

All'interno della routine di gestione degli errori, viene usata l'istruzione RESUME che effettua le seguenti operazioni

1. elimina la condizione di errore in atto

2. esegue un salto incondizionato (GOTO) ad un determinato punto del programma

A seconda di dove deve riprendere l'esecuzione del programma dopo l'errore, la sintassi di RESUME varia di conseguenza; essa assume le seguenti forme

RESUME [**0**] semplicemente l'istruzione o seguita dal numero 0, se si desidera che l'esecuzione riprenda dall'istruzione che ha causato l'errore; questo è necessario, per esempio,

quando si manifesta un errore nell'uso della stampante; infatti, dopo che si rimedia all'errore è necessario ripetere l'istruzione di stampa alla quale era stato generato l'errore;

RESUME {**numlinea** | **etichetta**} usando un numero di linea o un'etichetta dopo l'istruzione RESUME, il controllo del programma riprende dal punto indicato dopo l'errore.

Non è possibile specificare un'istruzione RESUME senza che una routine di gestione degli errori attiva.

Nel seguente esempio si può notare come, con una corretta gestione degli errori, viene garantita una veste professionale ad ogni programma scritto in Quick Basic.

```
ON ERROR GOTO ERTRAP
STARTPOINT:         ' Inizio del programma
        …           ' Corpo del
        …           ' programma
END
ERTRAP:
        PR=CSRLIN
        PC=POS(0)
        SELECT CASE ERR
                CASE 24,25
                        LOCATE 24,10
                        PRINT "STAMPANTE NON COLLEGATA"
                CASE 27
                        LOCATE 24,10
                        PRINT "CARTA ESAURITA"
                CASE ELSE
                        ON ERROR GOTO 0
        END SELECT
        PRINT "… (ESC=RINUNCIA - RET=CONTINUA)";
        BEEP
        A$=" "
        DO WHILE ASC(A$)<>27 AND ASC(A$)<>13
                A$=INPUT$(1)
        LOOP
        LOCATE 24,10
        PRINT SPACE$(60);
        LOCATE PR,PC
        IF ASC(A$)=13 THEN
                RESUME
        ELSE
                RESUME STARTPOINT
        END IF
```

*** ON GOSUB**

Vedere ON GOTO

*** ON GOTO**

Capita spesso che, a seconda del valore assunto da una variabile, si debba eseguire una parte di programma piuttosto che un'altra; a questo scopo sono utilizzabili le istruzioni ON GOTO e ON GOSUB le cui sintassi sono le seguenti

ON esprnum GOTO {listanumlinea | listaetichette}

ON esprnum GOSUB {listanumlinea | listaetichette}

in cui esprnum è una espressione numerica il cui valore deve essere intero positivo compreso tra 0 e 255 (limiti inclusi); per i valori negativi e quelli maggiori di 255 viene generato un errore. Ad ogni valore assunto da esprnum può corrispondere un riferimento (numero di linea od etichetta) nella lista seguente l'istruzione GOSUB o GOTO; l'esecuzione del programma riprenderà al riferimento associato ad esprnum. Nel caso in cui venga usata l'istruzione GOSUB, al termine della sottoroutine (dopo l'esecuzione dell'istruzione RETURN), il controllo ritorna all'istruzione seguente l'ON GOSUB.

Se esprnum assume un valore nullo o maggiore del numero di riferimenti disponibili, l'esecuzione del programma continua con l'istruzione seguente. Il prossimo esempio mostra l'uso dell'istruzione ON GOTO

```
CLS
DO
    INPUT “IMMETTERE UN NUMERO TRA 1 E 5 “,V
    ON V GOTO PRI,SEC,TER,QUA,QUI
    PRINT “NUMERO NON VALIDO …”
LOOP
PRI:
    PRINT “UNO”
    END
SEC:
    PRINT “DUE”
    END
TER:
    PRINT “TRE”
    END
QUA:
    PRINT “QUATTRO”
    END
QUI:
    PRINT “CINQUE”
    END
```

È consigliabile usare la struttura di controllo SELECT CASE..END SELECT al posto della ON GOTO e della ON GOSUB perché la prima si dimostra più flessibile e potente.

*** ON KEY(n)**

Con questa istruzione è possibile eseguire una sottoroutine utente dopo che è stato premuto un tasto o una combinazione di tasti definita in precedenza. La sintassi dell'istruzione è la seguente

ON KEY(n) GOSUB {**numriga** | **etichetta**}

in cui 'n' è il numero del tasto da rilevare, 'numriga' o 'etichetta' è il riferimento definito dall'utente, dal quale inizia la sottoroutine.
Il numero del tasto 'n' è riferito a quello definito con l'istruzione KEY a cui si rimanda per ulteriori spiegazioni.

Per eseguire dei programmi che usino questa istruzione è necessario specificare al compilatore di linea BC, l'opzione /v o l'opzione /w per abilitare la rilevazione degli eventi; l'opzione /v viene specificata per una rilevazione più continua degli stessi; se si crea un file EXE dall'ambiente, tale parametro viene automaticamente incluso.

Ad esempio, il prossimo programma mostra come è possibile intercettare l'uso dei tasti Ctrl Alt e Del durante l'escuzione dello stesso, per evitare il reset del computer

```
' (Main Module)
CLS
KEY 15, CHR$(&HC)+CHR$(&H53)
ON KEY(15) GOSUB CTRLALTDEL
KEY(15) ON
PRINT "PREMERE IL TASTO F PER TERMINARE …"
DO
LOOP UNTIL INKEY$="F"
END
CTRLALTDEL:
  PR=CSRLIN
  PC=POS(0)
  LOCATE 24,5
  COLOR 15
  PRINT "- CTRL ALT E DEL SONO STATI PREMUTI -";
  PRINT " RET=CONFERMA  ESC=RINUNCIA -";
  DO
        SOUND 500,2
        A$=INPUT$(1)
  LOOP UNTIL INSTR(CHR$(13)+CHR$(27), A$) > 0
  IF A$=CHR$(27) THEN
        LOCATE 24,5
        PRINT SPACE$(66);
        COLOR 7
        LOCATE PR,PC
        RETURN
  ELSE
        SOUND 800,2
        CLS
```

```
            DEF SEG=&HFFFF
            K%=0
            CALL ABSOLUTE(K%)
        END IF
```

Nella prima istruzione KEY, è stato definito il tasto numero 15 (primo tasto utente) con i valori esadecimale C e 53 che corrispondono, rispettivamente, alla combinazione dei tasti Ctrl e Alt (codici 4 + 8 decimali) e al tasto Del del tastierino numerico (codice 83 decimale); per i codici dei tasti vedere le tabelle riportate all'istruzione KEY. Le due linee seguenti, contengono le istruzioni per la definizione della sottoroutine da eseguire e per l'abilitazione della funzione. Da quel punto inizia il programma dell'utente, ridotto in quest'esempio, alle istruzioni per l'attesa del tasto F.
La routine che controlla l'uso del tasto definito, inizia salvando la posizione del cursore in due variabili numeriche e ponendo all'operatore, nella linea 24 del video, la domanda per la scelta dell'operazione da compiere dopo l'uso dei tasti Ctrl Alt e Del; infatti, a questo punto, è possibile, tramite il tasto ESC, rinunziare al reset e continuare il programma dopo avere eliminato la frase di richiesta dal video ed avere riposizionato il cursore nella sua posizione precedente; oppure, tramite il tasto Return, si può confermare l'operazione eseguendo con l'istruzione CALL ABSOLUTE, le istruzioni contenute nella ROM del BIOS ad iniziare dall'indirizzo assoluto FFFF:0000 (in esadecimale) che corrisponde al punto di reset del processore delle serie i8088, i80x86. Per compilare il programma da DOS è necessario dare la seguente linea

BC /v ESK;

in cui l'opzione /v è usata per la rilevazione degli eventi. Per quanto riguarda la creazione del file EXE da DOS, è necessaria la linea seguente

LINK ESK,,NUL,QB.LIB

in cui viene specificata la libreria in cui trovare il modulo ABSOLUTE; questo modulo, per la versione 4.5, è già incluso nella libreria standard.
Dall'ambiente, tramite il menu Run opzione Make EXE…, il programma viene compilato correttamente da Quick Basic, a condizione che venga caricata la libreria QB.QLB all'apertura con il comando

QB ESK /LQB

anche per la versione 4.5 del linguaggio.

Attenzione al fatto che la combinazione di tasti controllata è **solo** quella specificata e che lo stato della tastiera influisce sul suddetto controllo. Infatti, se il precedente programma viene eseguito, **non devono essere attivi** il Num Lock né il Caps Lock; lo stesso se si premesse anche un tasto Shift insieme alla combinazione di tasti controllata. Per eseguire un completo controllo della combinazione Ctrl Alt Del, bisognerebbe considerare anche tutti i casi in cui si può trovare la tastiera quando i suddetti tasti sono premuti; ad esempio, per includere anche il controllo con il tasto Caps Lock attivato, bisogna includere le seguenti linee nel programma precedente

```
KEY 16,CHR$(&HC + &H40)+CHR$(&H53)        ' &H40=Caps Lock attivo
```

```
ON KEY(16) GOSUB CTRLALTDEL
KEY(16) ON
```

Considerare anche il fatto che, per la tastiera di tipo avanzato, per usare il tasto Delete che non è nel tastierino, bisogna avvalersi del valore &H80 nel primo carattere del tasto definito.

*** ON PEN**

Questa istruzione è necessaria per specificare una routine per la gestione degli eventi determinati dalla penna ottica. La sua sintassi è

ON PEN GOSUB {**numriga** | **etichetta**}

in cui 'numriga' o 'etichetta' è il riferimento definito dall'utente, dal quale inizia la sottoroutine.

In seguito con l'istruzione

PEN {**ON** | **OFF** | **STOP**}

è possibile intercettare degli eventi che si verificano in modo asincrono e che interessano la penna ottica. La sua sintassi prevede un termine scelto tra ON, OFF e STOP

PEN ON abilita l'intercettazione degli eventi che si riferiscono alla penna ottica, tramite una routine definita dall'istruzione ON PEN GOSUB. Dopo aver specificato il termine ON, se un evento è rilevato, viene eseguita dunque, la routine specificata dopo il GOSUB

PEN OFF disabilita la funzione di intercettazione degli eventi riguardanti la penna ottica; la routine specificata dopo il GOSUB (nell'istruzione ON PEN..) non viene più eseguita e gli eventi eventualmente accaduti, vengono ignorati

PEN STOP agisce come la precedente opzione, ma gli eventi accaduti, nei limiti del buffer a disposizione, sono conservati per potere eventualmente riprendere la loro elaborazione, tramite la PEN ON, in un secondo momento; questa opzione viene utilizzata per sospendere temporaneamente per brevi periodi, l'elaborazione dei dati ricevuti.

In generale, lo scheletro del programma tipo che usa tale istruzione per la penna ottica, è la seguente

```
ON PEN GOSUB PENEVENT
PEN ON          ' Gli eventi sono rilevati
…
…
PEN STOP        ' Gli eventi non sono rilevati ma conservati
…
…
PEN OFF         ' Gli eventi non sono rilevati
END
PENEVENT:
    …           ' Routine trattamento eventi
```

```
…                    ' penna ottica
RETURN
```

*** ON PLAY(n)**

È l'istruzione usata per eseguire una sottoroutine quando il numero di note da suonare presenti nel buffer apposito, diventa minore di un numero 'numnote' specificato come argomento. La sua sintassi è la seguente

ON PLAY(numnote) GOSUB {**numlinea** | **etichetta**}

La sottoroutine viene eseguita quando il numero di note presenti nel buffer varia da numnote a numnote-1. L'intervallo di valori possibili è tra 1 e 32.
Questa istruzione permette il salto a tale sottoroutine solo se la musica è stata impostata in background (sottocomando MB dell'istruzione PLAY); inoltre, il salto a tale routine avviene **soltanto** quando il numero di note passa da numnote a numnote-1 e in nessun altro caso, anche se le note nel buffer sono in numero minore di numnote.

La precedente istruzione è usata **solamente** per definire la sottoroutine che deve essere eseguita ad ogni evento; ma, per abilitare o disabilitare tale esecuzione è inoltre necessaria, quest'altra istruzione, la cui sintassi è

PLAY {**ON** | **OFF** | **STOP**}

Tramite questa è possibile intercettare l'evento che si verifica in modo asincrono. La sua sintassi prevede un termine scelto tra ON, OFF e STOP

PEN ON abilita l'intercettazione degli eventi quando il numero delle note che sono nel buffer, varia dal numero indicato ad uno in meno, tramite una routine definita dall'istruzione ON PLAY GOSUB. Dopo aver specificato il termine ON, se un evento è rilevato, viene eseguita dunque, la routine specificata dopo il GOSUB

PEN OFF disabilita la funzione di intercettazione degli eventi suddetti; la routine specificata dopo il GOSUB (nell'istruzione ON PLAY..) non viene più eseguita e gli eventi eventualmente accaduti, vengono ignorati

PEN STOP agisce come la precedente opzione, ma gli eventi accaduti, nei limiti del buffer a disposizione, sono conservati per potere eventualmente riprendere la loro elaborazione, tramite la PLAY ON, in un secondo momento; questa opzione viene utilizzata per sospendere temporaneamente per brevi periodi, l'elaborazione degli eventi che si sono verificati.

Quando si verifica un evento e viene richiamata la sottoroutine apposita, viene implicitamente eseguito un PLAY STOP ed alla fine della stessa, un PLAY ON per evitare delle chiamate ricorsive della sottoroutine stessa.

Un programma che contiene tali istruzioni, se viene compilato sotto MS-DOS con BC, richiede la specifica nella linea di comando, del parametro **/v** o **/w**.

Il seguente programma di esempio, mostra l'uso della ON PLAY… e della ON TIMER…; esso rappresenta un piccolo esempio (molto spartano, in verità) di realizzazione di un video-game, ma contribuisce a comprendere meglio il funzionamento delle istruzioni suddette.

```
' (Main Module)
ON TIMER(1) GOSUB TEMPO
ON PLAY(2) GOSUB SUONA
TIMER ON
PLAY ON
CLS
GOSUB SUONA
XP=2
YP=2
XSP=1
YSP=1
XC=40
SHOT=0
BOOM=0
TB=2
SC=0
TI=120
LOCATE 25,20
PRINT USING "Time : ###    Score : ######"; TI; SC;
DO
        LOCATE YP,XP
        PRINT "O";
        IF XP=80 OR XP=1 THEN
                XSP=-XSP
        END IF
        IF YP=15 OR YP=1 THEN
                YSP=-YSP
        END IF
        FOR DP=1 TO 100
        NEXT DP
        LOCATE YP,XP
        PRINT " ";
        XP=XP+XSP
        YP=YP+YSP
        LOCATE 23,XC
        PRINT " | ";
        LOCATE 24,XC
        PRINT "–";
        R$=RIGHT$(INKEY$, 1)
        IF R$=CHR$(75) THEN
                LOCATE 23,XC
                PRINT "  ";
                LOCATE 24,XC
                PRINT "  ";
```

```
        IF XC>2 THEN
                XC=XC-2
        END IF
END IF
IF R$=CHR$(77) THEN
        LOCATE 23,XC
        PRINT "  ";
        LOCATE 24,XC
        PRINT "  ";
        IF XC<78 THEN
                XC=XC+2
        END IF
END IF
IF R$=CHR$(32) THEN
        IF NOT SHOOT THEN
                SHOOT=-1
                XXS=XC+1
                YYX=23
        END IF
END IF
IF SHOT THEN
        LOCATE YYS+1,XXS
        PRINT " ";
        LOCATE YYS,XXS
        PRINT "*";
        YYS=YYS-1
        IF YYS=0 THEN
                SHOT=0
                LOCATE YYS+1,XXS
                PRINT " ";
        END IF
        IF(ABS(XXS-XP)<2) AND (ABS(YYS-YP)<2) THEN
                PLAY OFF
                BEEP
                XBO=XP-2
                YBO=YP+1
                LOCATE YBO,XBO
                PRINT "BOOM";
                BOOM=-1
                SC=SC+10*(2-ABS(YYS-YP))
          SC=SC+10*(2-ABS(XXS-XP))
                LOCATE 25,36
                PRINT USING "Score : ######"; SC;
                SHOT=0
                PLAY ON
                GOSUB SUONA
                LOCATE YYS+1,XXS
                PRINT " ";
```

```
                YP=2
                XP=2
            END IF
        END IF
LOOP
END
SUONA:
PLAY "MB L16 CCDCFE L16 N0"
RETURN
TEMPO:
TI=TI-1
OX=POS(0)
OY=CSRLIN
LOCATE 25,20
PRINT USING "Time : ###"; TI;
IF TI=0 THEN
        TIMER OFF
        PLAY OFF
        LOCATE 12,35
        PRINT "*** GAME OVER ***"
        A$=INPUT$(1)
        END
END IF
IF BOOM THEN
        TB=TB-1
        IF TB=0 THEN
            LOCATE YBO,XBO
            PRINT "   ";
            BOOM=0
            TB=2
        END IF
END IF
LOCATE OY,OX
RETURN
```

Per compilare sotto MS-DOS tale programma, sono sufficiente le seguenti linee di comando

BC /V /W GAME;

LINK GAME;

Viene così creato il file GAME.EXE che può essere eseguito da MS-DOS direttamente.

*** ON STRIG(n)**

Tramite questa istruzione si specifica una routine per la gestione degli eventi determinati dall'uso di uno dei tasti di un joystick. La sua sintassi è

ON STRIG(codtasto) GOSUB {**numriga** | **etichetta**}

in cui ‘numriga’ o ‘etichetta’ è il riferimento definito dall’utente, dal quale inizia la sottoroutine, e ‘codtasto’ è il codice del tasto che, quando usato, esegue la routine; tale codice è un valore numerico intero scelto secondo la seguente tabella

	Joystick 1	Joystick 2
Tasto inferiore	**0**	**2**
Tasto superiore	**4**	**6**

Inoltre, con l’istruzione

STRIG {**ON** | **OFF** | **STOP**}

è possibile intercettare degli eventi che si verificano in modo asincrono e che interessano i pulsanti dei joysticks. La sua sintassi prevede un termine scelto tra ON, OFF e STOP

STRIG ON abilita l’intercettazione degli eventi che si riferiscono ai pulsanti dei joysticks, tramite una routine definita dall’istruzione ON STRIG GOSUB. Dopo aver specificato il termine ON, se un evento è rilevato, viene eseguita dunque, la routine specificata dopo il GOSUB

STRIG OFF disabilita la funzione di intercettazione degli eventi riguardanti i joysticks; la routine specificata dopo il GOSUB (nell’istruzione ON STRIG..) non viene più eseguita e gli eventi eventualmente accaduti, vengono ignorati

STRIG STOP agisce come la precedente opzione, ma gli eventi accaduti, nei limiti del buffer a disposizione, sono conservati per potere eventualmente riprendere la loro elaborazione, tramite la STRIG ON, in un secondo momento; questa opzione viene utilizzata per sospendere temporaneamente per brevi periodi, l’elaborazione degli eventi che si sono verificati.

In generale, lo scheletro del programma tipo che usa tale istruzione per il trattamento degli eventi dei joysticks, è il seguente

```
ON STRIG(0) GOSUB JOY1LOW
ON STRIG(2) GOSUB JOY1UPP
ON STRIG(4) GOSUB JOY2LOW
ON STRIG(6) GOSUB JOY2UPP
STRIG ON              ‘ Gli eventi sono rilevati
…
…
STRIG STOP            ‘ Gli eventi non sono rilevati ma conservati
…
…
STRIG OFF             ‘ Gli eventi non sono rilevati
END
JOY1LOW:
```

```
        J=1
        GOTO JOY
JOY1UPP:
        J=2
        GOTO JOY
JOY2LOW:
        J=3
        GOTO JOY
JOY2UPP:
        J=4
JOY:
        SELECT CASE J
                CASE 1
                        …       ' Tasto inferiore joystick 1
                CASE 2
                        …       ' Tasto superiore joystick 1
                CASE 3
                        …       ' Tasto inferiore joystick 2
                CASE ELSE
                        …       ' Tasto superiore joystick 2
        END SELECT
        RETURN
```

*** ON TIMER(n)**

Questa istruzione è usata per definire una sottoroutine che viene eseguita ogni 'numsecondi', valore che viene specificato come argomento tra parentesi. La sintassi della ON TIMER.. è

ON TIMER(numsecondi) GOSUB {**numriga** | **etichetta**}

in cui, dopo la parola chiave GOSUB, viene specificato il riferimento alla sottoroutine. L'intervallo dei valori da potere attribuire a 'numsecondi', va da 1 a 86400 (da 1 secondo a 24 ore) e non sono ammessi valori decimali.
Come per tutte le istruzioni simili, la rilevazione dell'evento deve essere abilitato tramite l'istruzione apposita che, in questo caso, ha la seguente sintassi

TIMER {**ON** | **OFF** | **STOP**}

La sua sintassi prevede un termine scelto tra ON, OFF e STOP

TIMER ON abilita l'intercettazione degli eventi quando è passato il numero di secondi specificato nella ON TIMER(n). Dopo aver specificato il termine ON, se è passato il tempo specificato, viene eseguita dunque, la routine specificata dopo il GOSUB

TIMER OFF disabilita la funzione di intercettazione degli eventi suddetti; la routine specificata dopo il GOSUB (nell'istruzione ON TIMER..) non viene più eseguita e gli eventi eventualmente accaduti, vengono ignorati

TIMER STOP agisce come la precedente opzione, ma gli eventi accaduti, nei limiti del buffer a disposizione, sono conservati per potere eventualmente riprendere la loro elaborazione, tramite la TIMER ON, in un secondo momento; questa opzione viene utilizzata per sospendere temporaneamente per brevi periodi, l'elaborazione degli eventi che si sono verificati.

È possibile vedere un esempio di utilizzo di tali istruzioni nell'esempio fornito per l'istruzione ON PLAY, ma il seguente piccolo programma dà un ulteriore suggerimento per l'uso della ON TIMER

```
ON TIMER(1) GOSUB OROLOGIO
TIME$=”12:00”
TIMER ON
CLS
LOCATE 10,10
PRINT “ESC PER TERMINARE…”
DO
LOOP WHILE INKEY$<>CHR$(27)
END
OROLOGIO:
  OX=POS(0)
  OY=CSRLIN
  LOCATE 1,60
  PRINT MID$(DATE$, 4, 3); LEFT$(DATE$, 3); RIGHT$(DATE$, 4);
  PRINT “ “; TIME$;
  LOCATE OY,OX
  RETURN
```

*** OPEN**

L'istruzione OPEN è necessaria prima della lettura o scrittura di un file o di una periferica valida. È infatti l'istruzione che provvede a generare il buffer per l'I/O, definire le modalità di accesso al file e regolare la condivisione dello stesso in ambiente di rete (solo con DOS versione 3.0 e seguenti, con il programma SHARE del DOS installato correttamente).

L'istruzione OPEN ha due sintassi diverse di cui, la prima, è stata mantenuta per compatibilità con versioni precedenti di Basic, mentre la seconda, è quella completa con cui è possibile sfruttare le possibilità di gestione di rete locale del DOS (dalla versione 3.0 in poi);

prima sintassi:

OPEN modo1, [**#**]**nfile, file,** [**,lrec**]

seconda sintassi:

OPEN file [**FOR modo2**][**ACCESS acc**][**blk**] **AS** [**#**]**nfile** [**LEN=lrec**]

In ambedue le sintassi il numero 'nfile' preceduto dal simbolo opzionale #, indica il numero a cui ci si dovrà riferire in seguito per agire sul file aperto; tale numero è compreso tra 1 e 255 e deve essere unico per ogni file aperto in un determinato momento; se il file viene chiuso, il numero può essere utilizzato nuovamente da un altro file.

Il parametro ‘file’, invece, è una espressione stringa che specifica il nome di un file, eventualmente completo di percorso e drive, e che segue le regole imposte dal DOS per la sua formazione; sono validi quindi, i seguenti esempi

“A:\PROVA.DAT” “ESEMPIO” “D:\QB40\DATA\PROG.DAT”

mentre non lo sono questi altri

“AA:PROVA.DAT” “ESEMPIO:DAT” “D:\QB40 PROG.DAT”

e tutti quelli in cui il percorso, anche se scritto correttamente, non è giusto perché una o più sottodirectory non esistono. Dato che ‘file’ è una espressione stringa, è possibile specificare, per esempio, il nome del file anche nel seguente modo

DR$ + PH$ + FI$

in cui le tre variabili stringa contengono, rispettivamente il drive, il percorso e il nome del file da aprire.
Inoltre, si può specificare in questo parametro un nome di periferica tra le seguenti

CONS: SCRN: KYBD: LPTn: COMn:

in cui, le prime due (CONS: e SCRN:) si riferiscono al video e sono accessibili solo in uscita; la seguente (KYBD:) è riferita alla tastiera ed è accessibile in input; la prossima (LPTn:) è una delle stampanti parallele collegabili il cui numero è il valore ‘n’ (LPT1: LPT2: LPT3:), e l’accesso consentito è in scrittura, tranne che non venga specificata l’ozione BIN (LPT1:BIN) per permettere l’accesso binario; l’ultima, infine, si riferisce ad una delle porte seriali, a seconda del valore ‘n’ (COM1: COM2:) ed, in questo caso l’accesso è possibile in lettura-scrittura; in tutti i casi, l’accesso non è bufferizzato.
La periferica LPTn: è sempre controllabile tramite i comandi LPRINT e WIDTH LPRINT, mentre, per le porte seriali, in seguito verrà descritta con maggiori dettagli la particolare forma di OPEN correlata (OPEN “COMn”), che permette l’uso di buffer per le operazioni di I/O.

La modalità di accesso usata nella seconda sintassi (modo2) è una delle seguenti

INPUT modalità di input sequenziale; è possibile aprire, con lo stesso nome, più di una volta lo stesso file con numeri di riferimento diversi

OUTPUT modalità di output sequenziale; prima di aprire un’altra volta lo stesso file, anche se con numeri di riferimento diversi, è necessario chiudere quello aperto per prima

APPEND modalità di output sequenziale in accodamento; posiziona il puntatore del file alla fine di questo per permettere ai dati scritti con PRINT # e WRITE # di essere aggiunti a quelli esistenti; anche per questa modalità non è possibile aprire lo stesso file più volte contemporaneamente

RANDOM modalità di accesso casuale; è la modalità standard adottata da Quick Basic se altre non sono specificate; l'accesso è in scrittura/lettura ma, se questo non è possibile, in sola scrittura o sola lettura; in presenza della opzione ACCESS è quest'ultima a decidere il tipo di accesso

BINARY modalità di accesso binario; in questo modo è possibile leggere e scrivere nel file aperto a livello di singolo byte; senza clausola ACCESS, l'accesso è in lettura/scrittura o, se questo non è possibile in sola scrittura o lettura

Nella prima sintassi, queste modalità di apertura del file, sono specificati dal parametro modo1, che è una espressione stringa, semplicemente con l'iniziale della modalità

I Input sequenziale
O Output sequenziale
A Modo Append
R Modo Random
B Modo Binary

Il parametro 'lrec' nelle due sintassi, si riferisce alla lunghezza del buffer predisposto per l'I/O del record; questo valore può essere compreso tra 1 e 32767 e se il file è aperto in modo random, determina la lunghezza del record che deve corrispondere o essere maggiore della somma delle lunghezze dei campi espresse nell'istruzione FIELD; se manca, viene assunto per tali file il valore di 128 bytes. Per i file sequenziali, è la lunghezza del buffer di I/O usato per il trasferimento dei dati e non corrisponde alla lunghezza del record dato che questa è variabile; un valore elevato sveltisce le operazioni di I/O ma comporta l'uso di una quantità di memoria superiore; il valore di default di 512 bytes è quasi sempre ottimale. Infine, per i file aperti in modo BINARY, tale parametro non è utilizzato e, se presente, ignorato.

Solo nella seconda versione dell'istruzione OPEN, è possibile specificare due parametri, la modalità di accesso e quella di bloccaggio, indispensabili quando si lavora in rete locale; per questo uso, è indispensabile usare una versione del DOS eguale o successiva alla 3.0 e bisogna installare in precedenza, il modulo SHARE.EXE che regola l'accesso di diversi processi a dati contenuti in file comuni.
Dopo la parola chiave ACCESS, è possibile specificare le seguenti combinazioni di clausole, permettendo la definizione delle operazioni consentite sul file aperto

READ	file aperto per la sola lettura
WRITE	file aperto per la sola scrittura
READ WRITE	file aperto per le due modalità; è valida solo per i modi RANDOM, BINARY ed APPEND

Se un file è già aperto da un programma ed un altro lo apre in una modalità non consentita, Quick Basic emette un messaggio di errore appropriato (Access denied).
Il parametro 'blk' invece, sempre in multiutenza, regola l'accesso di diversi programmi al file interessato; le seguenti combinazioni di parametri sono valide

SHARED il file è condiviso con tutti i programmi che lo vogliono aprire. Qualsiasi processo può leggere e scrivere il file aperto

LOCK READ il file non può essere letto da nessun altro programma che lo richieda in seguito; se già qualche processo ha aperto in lettura il file, questa modalità di blocco non viene concessa

LOCK WRITE il file non può essere scritto da nessun altro programma che lo richieda in seguito; se già qualche processo ha aperto in scrittura il file, questa modalità di blocco non viene concessa

LOCK READ WRITE con questa modalità, l'accesso al file viene completamente riservato ad un processo a cui viene permessa sia la lettura che la scrittura dello stesso. Se un altro processo ha ottenuto in precedenza un qualsiasi tipo di accesso al file considerato, questo blocco non è concesso.

Se questa clausola manca, il processo corrente può aprire il file in lettura e scrittura un qualsiasi numero di volte, ma altri non possono farlo.

Ad esempio, dopo avere installato il modulo SHARE.EXE usando almeno il DOS versione 3.00, eseguire il seguente programma

```
OPEN "PROVA" FOR BINARY ACCESS WRITE LOCK WRITE AS #1
OPEN "PROVA" FOR BINARY ACCESS READ LOCK READ AS #2
S$=SPACE$(100)
PUT #1,1,S$
GET #2,1,S$
CLOSE
```

in cui il file numero 1, viene aperto in scrittura e tale modalità viene riservata (ACCESS WRITE LOCK WRITE), mentre lo stesso, con il numero 2, viene aperto in lettura e bloccato (ACCESS READ LOCK READ); il file può quindi essere letto solo tramite il canale 2 e scritto solo tramite il canale 1; altri processi non possono accedere al file.
Nel prossimo esempio, invece, il file non è bloccato da nessuna frase di apertura anzi è condiviso da tutte e l'accesso è, per la prima, solo in scrittura, per la seconda solo in lettura e per la terza è libero

```
OPEN "ESEMPIO" FOR BINARY ACCESS WRITE SHARED AS #1
OPEN "ESEMPIO" FOR BINARY ACCESS READ SHARED AS #2
OPEN "ESEMPIO" FOR BINARY ACCESS READ WRITE SHARED AS #3
S$=SPACE$(100)
PUT #1,1,S$
GET #2,1,S$
PUT #3,1,S$
GET #3,1,S$
CLOSE
```

Ecco di seguito un esempio di lettura e scrittura da file sequenziale che si suppone già esistente

```
OPEN "FILE1" FOR INPUT AS #1
```

```
OPEN "FILE2" FOR OUTPUT AS #2
DO WHILE NOT EOF(1)
        INPUT #1,A$
        PRINT #2,A$
LOOP
CLOSE
```

e la lettura scrittura di un file random

```
OPEN "FILEA" FOR RANDOM AS #1 LEN=100
OPEN "FILEb" FOR RANDOM AS #2 LEN=100
FIELD #1, 100 AS FA$
FIELD #2, 100 AS FB$
NR=LOF(1)/100
DO WHILE NR>0
        GET #1,NR
        LSET FB$=FA$
        PUT #2,NR
        NR=NR-1
LOOP
CLOSE
```

*** OPEN "COM"**

Quando l'istruzione OPEN, descritta in precedenza, viene usata per aprire un canale seriale tramite il nome di periferica COM, allora si devono specificare un numero di parametri tanto alto da doverla esaminare in particolare. La sintassi dell'istruzione OPEN in questo caso, assume la seguente forma

OPEN "COMn:[**v**][**,**[**p**][**,**[**d**][**,s**]]]][**lstopz**]**"** [**FOR m**] **AS** [**#**]**f** [**LEN=lr**]

in cui il numero n dopo la parola COM si riferisce al numero di porta seriale da aprire; normalmente, possono essere installate COM1 e COM2 tra cui scegliere, ma esistono schede per potere collegare anche 8 e più interfacce seriali al sistema; bisogna essere sicuri che la porta sia installata prima di aprirla. I parametri seguenti, assumono i seguenti significati e valori

v è la velocità di trasmissione assegnata alla porta ed è espressa in baud (bits per secondo); i valori validi sono 75, 110, 150, 300, 600, 1200, 1800, 2400, 9600. Per le comunicazioni via modem, le velocità più usate sono 300, 600, 1200 e 2400. La velocità di 9600 può essere usata per comunicazioni seriali via cavo a breve distanza. Il valore usato per default è 300

p è la convenzione di parità adottata dai sistemi in collegamento; la parità è un modo con cui possono essere rilevati degli errori di trasmissione o di ricezione di dati ed è fondamentale che i due sistemi, il trasmittente e il ricevente, adottino la stessa modalità; la parità può assumere i seguenti valori: N (parity None), E (Even parity), O (odd parity), S (Space parity), M (Mark parity); negli ultimi due casi viene mandato un bit costante di parità, 0 o 1 rispettivamente; la parità N (nessuna parità) è necessaria se si adottano 8 bits per i dati in trasmissione. Per default è adottata la parità Even

d rappresenta il numero dei bits che compongono ogni carattere che si vuole trasmettere; può essere eguale a 5, 6, 7, oppure 8; normalmente questi ultimi due valori sono i più usati. Il valore usato per default è 7

s è il numero di bits di stop usati dal protocollo di comunicazione; infatti questo prevede, per ogni carattere trasmesso, 1 bit di start, i bits che compongono il dato, il bit di parità se presente ed un numero variabile di bits di stop che marcano la fine del carattere trasmesso. Il valore da adottare in trasmissione dipende da quello adottato dal ricevente; l'importanza è che i valori usati dai due sistemi siano eguali per una corretta sincronizzazione. Il valore di default è 1.

A questi valori segue una lista di parametri che controllano l'handshake; per ogni valore del primo gruppo che non è usato, deve essere usata una virgola. Nella listadi parametri che segue, possono essere specificate le seguenti opzioni, in qualsiasi ordine separate da virgole

> **ASC** viene aperto il canale di comunicazione in modalità ASCII. In questo modo le tabulazioni sono espanse (convertite in spazi), viene inserito il LF alla fine di ogni linea di caratteri trasmessa ed, alla chiusura della trasmissione viene emesso un carattere di EOF (1A esadecimale)
>
> **BIN** in alternativa alla precedente modalità, il canale di comunicazione viene aperto in modo binario. Non viene eseguita alcuna operazione sui dati trasmessi né vengono trasmessi automaticamente caratteri di segnalazione. È il modo adottato per default
>
> **CD**[**n**] con questo parametro, opzionalmente seguito da un valore, si imposta il controllo di timeout di periferica. Tale errore viene evidenziato se la linea DCD (Data Carrier Detect) è a 0 per più di n millisecondi. Questo valore è per default 0 e può essere compreso tra 0 e 65535; con 0, il controllo viene ignorato
>
> **CS**[**n**] anche questo parametro, seguito da un valore opzionale, controlla il timeout di periferica. Se la linea CTS (Clear To Send) è a 0 per più di n millisecondi, interviene l'errore. Per default, questo valore è 1000 (1 secondo) e può essere compreso tra 0 e 65535; con 0 il controllo viene ignorato
>
> **DS**[**n**] è il parametro che controlla il timeout sulla linea DSR (Data Set Ready). Se questa linea è a 0 per più di n millesimi di secondo, interviene l'errore. Il valore è, per default, 1000 (1 secondo) ma può variare tra 0 e 65535; con 0, il controllo viene ignorato
>
> **OP**[**n**] con questo parametro si prediponde il tempo massimo che l'istruzione OPEN deve attendere prima che si attui l'apertura del canale seriale; dopo tale tempo interviene un errore. Il valore può essere compreso tra 0 e 65535; il parametro OP usato senza valore permette che l'istruzione OPEN attenda al massimo 10 secondi mentre, se non viene usato, il tempo massimo corrisponde a 10 volte il valore massimo impostato per DS o CS

TB[**n**] imposta il numero di bytes usati nel buffer di trasmissione; il valore di default è di 512 e quello massimo di 32767

RB[**n**] imposta il numero di bytes usati nel buffer di ricezione; il valore di default è di 512 o quello specificato tramite l'opzione /C: all'apertura del Quick Basic o con il compilatore BC ed il valore massimo è 32767

RS elimina il controllo della linea RTS (Request To Send)

LF impostando tale opzione, viene automaticamente inserito un carattere di line-feed (0A esadecimale) in trasmissione dopo il carattere di carriage-return (0D esadecimale). È una opzione molto utile quando si intende usare una stampante collegata all'interfaccia seriale

Nell'istruzione OPEN il numero 'f' è quello a cui si riferiranno tutte le istruzioni di I/O per la comunicazione seriale; questo valore è compreso tra 1 e 255 e non deve essere usato contemporaneamente da un'altra frase OPEN.

Per quanto riguarda il modo di apertura 'm', questo può essere uno dei seguenti

INPUT solo ingresso dati da linea seriale
OUTPUT solo uscita dati su linea seriale
RANDOM lettura e scrittura sono consentite in maniera casuale; in questo caso il valore usato dopo la parola LEN si riferisce alla lunghezza del buffer associato per permettere l'uso di istruzione PUT E GET. Se questo valore manca, viene assunto per default 128.

Ad esempio, ecco lo scheletro di un programma che consente l'I/O da linea seriale emulando un terminale stupido (dumb)

```
' (Main Module)
DECLARE SUB HANDLEBACK (C$)
CLS
LOCATE 2,1
PRINT STRING$(80, 196);
LOCATE 1,6
COLOR 15
PRINT "Terminale RS-232";
PRINT TAB(38); "[Alt+Bcksp: termina] - [Alt+E: cambia eco]"
COLOR 7
LOCATE 1,27
PRINT "ECHO ON"
EC=-1
VIEW PRINT 3 TO 25
OPEN "COM1:300,N,8,1" FOR RANDOM AS #1 LEN=512
DO
    K$=INKEY$
    SELECT CASE K$
```

```
                CASE IS = CHR$(0)+CHR$(15)
                        VIEW PRINT
                        CLS
                        CLOSE
                        END
                CASE IS = CHR$(0)+CHR$(18)
                        PX=POS(0)
                        PY=CSRLIN
                        VIEW PRINT
                        LOCATE 1,32
                        EC=NOT(EC)
                        IF EC THEN
                                PRINT "ON ";
                                SOUND 800,1
                        ELSE
                                PRINT "OFF";
                                SOUND 200,1
                        END IF
                        VIEW PRINT 3 TO 25
                        LOCATE PY,PX
                CASE IS <> ""
                        PRINT #1,K$;
                        HANDLEBACK K$
                CASE ELSE
                        IF NOT EOF(1) THEN
                                M$=INPUT$(LOC(1), #1)
                                FOR C=1 TO LEM(M$)
                                        HANDLEBACK MID$(M$,C,1)
                                NEXT C
                        END IF
        END SELECT
LOOP
' (Sub HANDLEBACK)
SUB HANDLEBACK (K$) STATIC
        SHARED EC
        IF EC THEN
                IF ASC(K$)=8 THEN
                        X=POS(0)
                        IF X=1 THEN
                                X=80
                                LOCATE CSRLIN-1,X
                                PRINT " ";
                                LOCATE CSRLIN-1,X
                        ELSE
                                LOCATE  ,X-1
                                PRINT " ";
                                LOCATE  ,X-1
                        END IF
```

```
                ELSE
                        PRINT K$;
                END IF
        END IF
END SUB
```

*** OPTION BASE**

La semplice sintassi di questa frase di dichiarazione ammette soltanto un valore numerico come parametro; tale valore numerico deve essere costante e solamente 0 o 1

OPTION BASE {**0** | **1**}

è un parametro obbligatorio e serve ad indicare l'indice minimo per tutti gli arrays in cui non sia specificato altrimenti nell'istruzione DIM. Quest'ultima istruzione infatti, permette una dichiarazione più efficiente degli indici minimi e massimi di ogni arrays; la OPTION BASE è presente in Quick Basic solo per compatibilità con altre versioni di Basic.

Nel prossimo esempio, è da notare il fatto che l'array A ha l'elemento con indice 0 dichiarato esplicitamente nella frase DIM mentre, l'array B, ha indice minimo eguale a 1 per la frase dichiarativa iniziale e perché non lo ha esplicitamente dichiarato nella propria DIM

```
OPTION BASE 1
DIM A(0 TO 100), B(100)
```

L'uso di questa frase dichiarativa permette, molto spesso, di risparmiare memoria altrimenti occupata e non sfruttata; è infatti molto comune il fatto che, per gli arrays, gli elementi con un indice eguale a 0 non vengano sfruttati.

*** OR**

L'operatore logico OR esegue una operazione tra due valori interi o interi lunghi, bit per bit, rispettando la tavola della verità illustrata di seguito:

1^ Operando	2^ Operando	Risultato
0 (FALSE)	0 (FALSE)	0 (FALSE)
0 (FALSE)	**1 (TRUE)**	**1 (TRUE)**
1 (TRUE)	0 (FALSE)	**1 (TRUE)**
1 (TRUE)	**1 (TRUE)**	**1 (TRUE)**

L'operatore OR possiede, come gli altri operatori logici, il più basso livello di priorità in una espressione logico-aritmetica.

Vengono elencati di seguito due esempi che illustrano diversi modi di utilizzo dell'operatore suddetto:

```
CLS
INPUT A,B,C
IF B=0 OR C=0 THEN
```

```
        K=0
ELSE
        K=A/B*2/C
END IF
```

In questo caso, l'operatore di OR logico avviene tra i valori (B=0) e (C=0) che possono assumere solo i due risultati FALSE o TRUE.

```
CLS
DEF SEG=&H40
POKE &H17, PEEK(&H17) AND &HBF
DEF SEG
```

Al contrario dell'esempio mostrato per l'operatore AND, l'operatore OR è ora usato per porre ad 1 il bit numero 6 all'interno della locazione 0040:0017 in cui il DOS conserva lo stato del Caps Lock; dopo questa operazione, il modo di scrittura delle lettere diventa maiuscolo e la corrispondente spia nella tastiera, se presente, viene accesa. L'operazione effettuata è la seguente

```
Locazione 0040:0017      =    x X xxxxxx
Valore BF                =    01000000
Risultato in 0040:0017   =    x 1 xxxxxx
```

*** OUT**

Insieme alla funzione **INP**, questa istruzione è utilizzata nei programmi che sfruttano direttamente le risorse del sistema senza passare dalle routines e funzioni rese disponibili dal DOS e dalle BIOS. La sintassi dell'istruzione OUT è la seguente

OUT porta,dato

in cui il primo parametro è il numero della porta hardware di I/O verso la quale si vuole indirizzare il dato, fornito come secondo argomento. Sono molti i valori che possono essere indicati, per quanto riguarda il primo argomento; tutto dipende però, dal tipo di sistema e dalle sue caratteristiche di compatibilità.

I valori, minimi e massimi, usabili come argomenti, sono i seguenti

per la porta	da 0 a 65535 (0000 … FFFF esadecimale)
per il dato	da 0 a 255 (00 … FF esadecimale)

Il seguente esempio dimostra, con un sistema IBM compatibile, come è possibile sfruttare tutte le risorse hardware, tramite questa istruzione, in Quick Basic; in questo caso la porta 3F2h (esadecimale) corrisponde ad un registro di memoria interno del controller dei dischetti, tramile il quale, si riesce a controllare il 'led' (indicatore luminoso) di ogni drive installato

```
DECLARE SUB DELAY (SEC!)
DECLARE SUB DRIVERR (NV%, DR%, ACC!, SPE!)
CLS
```

```
DRIVERR 10, 1, .3, .3
END
SUB DELAY (SEC!)
    K=TIMER
    DO WHILE TIMER-K<SEC!
    LOOP
END SUB
DEFINT A-Z
SUB DRIVERR (NV, DR, ACC!, SPE!)
    TP1=&HF0 + DR – 1
    TP2=0
    IF DR=3 THEN
            TP1=&HF0
            TP2=&HF1
    END IF
    FOR V=1 TO NV
            OUT &H3F2, TP1
            DELAY ACC!
            SOUND 500,1
            OUT &H3F2, TP2
            DELAY SPE!
            SOUND 300,1
    NEXT V
    OUT &H3F2,0
END SUB
```

*** OUTPUT**

Vedere OPEN

*** PAINT**

È una delle istruzioni grafiche più potenti ed efficienti, usata per riempire una figura chiusa con un colore o un motivo di riempimento. La sua sintassi completa è la segunte

PAINT [STEP] (x,y) [,[interno][,[bordo][,sfondo]]]

La parola chiave STEP è opzionale e, se presente, fa interpretare le coordinate x e y come relative rispetto all'ultimo punto tracciato; altrimenti tali coordinate sono assolute, fisiche o logiche a seconda se sia o meno presente una istruzione WINDOW. Le coordinate x e y si riferiscono ad un punto **interno** alla figura da riempire; se tale punto cade all'esterno della figura stessa, questo verrà riempito mentre, se il punto è sul bordo, l'istruzione non avrà effetto.

Il parametro seguente le coordinate, anch'esso opzionale e sostituito dal colore di primo piano (foreground nell'istruzione COLOR) se non presente, può essere numerico o stringa. nel primo caso esso è il valore di un attributo a cui corrisponde un colore che deve essere usato per riempire l'area. La colorazione termina quando viene incontrato un colore che viene indicato come 'bordo'; usando, per questo parametro, il colore del bordo della figura da riempire, esso

viene usato come condizione per il termine dell'operazione di riempimento. Può essere omesso se il colore usato per il riempimento è lo stesso di quello usato per il bordo della figura.
Il seguente esempio mostra come è possibile colorare in maniera diversa, delle figure che hanno vario colore del bordo. I due valori, separati da virgole, richiesti dal programma, servono a riempire lo sfondo con il numero del colore dato per primo ed a limitare l'operazione con il colore fornito come secondo numero

```
DO
    CLS
    SCREEN 1
    CIRCLE (50, 50), 50, 1
    CIRCLE (250, 50), 50, 2
    CIRCLE (150, 100), 50, 3
    PAINT (50, 50), 3, 1
    PAINT (250, 50), 1, 2
    PAINT (150, 100), 2, 3
    LOCATE 20, 10
    INPUT C, B
    PAINT (0, 0), C, B
    A$=INPUT$(1)
LOOP WHILE A$<>CHR$(27)
SCREEN 0
END
```

Se per il parametro 'interno' si specifica una stringa invece di un valore numerico, questa viene intesa come un 'motivo' grafico da usare per riempire l'area indicata.
Questo 'motivo' viene formato su una griglia di dimensioni variabili, a seconda del modo grafico prescelto, ma che non può comunque avere più di 64 righe.

Nel modo grafico 2, che prevede una risoluzione di 640 x 200 con due soli colori, ad ogni pixel del motivo di riempimento corrisponde una colonna della griglia; ad esempio, per ricavare la stringa da specificare all'istruzione PAINT per ottenere il seguente motivo di riempimento

```
..x.....
.x.x....
x...x...
xxxxx...
x...x...
x...x...
........
```

è sufficiente interpretare ogni riga come se fosse un byte in cui i bits a 1 sono quelli che si vogliono visualizzare. Per la prima riga quindi, si avrà

..x..... → 00100000 → esadecimale 20

e per le altre

.x.x.... → 01010000 → esadecimale 50

x...x... → 10001000 → esadecimale 88
xxxxx... → 11111000 → esadecimale F8
x...x... → 10001000 → esadecimale 88
x...x... → 10001000 → esadecimale 88
.... → 00000000 → esadecimale 00

Il seguente programma mostra, infine, come è possibile sfruttare il precedente lavoro per riempire un cerchio con il motivo così definito

```
CLS
X$ = CHR$(&H20) + CHR$(&H50) + CHR$(&H88) + CHR$(&HF8)
X$ = X$ + CHR$(&H88) + CHR$(&H88) + CHR$(&H00)
SCREEN 2
CIRCLE (300, 100), 200
PAINT (300, 100), X$
A$ = INPUT$(1)
END
```

Invece, per riempire un'area con un motivo diverso dal precedente, ecco lo schema da realizzare con i conseguenti calcoli per ricavare la stringa da specificare nell'istruzione PAINT

x......x → 10000001 → esadecimale 81
.x....x. → 01000010 → esadecimale 42
..x..x.. → 00100100 → esadecimale 24
...xx... → 00011000 → esadecimale 18
...xx... → 00011000 → esadecimale 18
..x..x.. → 00100100 → esadecimale 24
.x....x. → 01000010 → esadecimale 42
x......x → 10000001 → esadecimale 81

Nel modo grafico 1, invece, ogni pixel può assumere uno tra 4 colori e quindi ad ogni pixel corrispondono 2 bits della matrice suddetta. Ad esempio, per realizzare un motivo di riempimento costituito da 4 strisce orizzontali dei 4 diversi colori, ecco lo schema già visto in precedenza

DDDD → 11111111 → esadecimale FF
BBBB → 01010101 → esadecimale 55
AAAA → 00000000 → esadecimale 00
CCCC → 10101010 → esadecimale AA

in cui le lettere A, B, C e D rappresentano i 4 colori che possono assumere i 4 pixels nella matrice considerando la seguente tabella di corrispondenza dei colori

colore A → 00
colore B → 01
colore C → 10
colore D → 11

Per le corrispondenze con i colori effettivi nel modo grafico 1, vedere le istruzioni SCREEN e COLOR. Se si volesse un motivo con le righe in verticale, ecco lo schema per il calcolo dei valori

DBAC	→	11010010	→	esadecimale D2
DBAC	→	11010010	→	esadecimale D2
DBAC	→	11010010	→	esadecimale D2
DBAC	→	11010010	→	esadecimale D2

o per un motivo a scacchi

BBAA	→	01010000	→	esadecimale 50
BBAA	→	01010000	→	esadecimale 50
DDCC	→	11111010	→	esadecimale FA
DDCC	→	11111010	→	esadecimale FA

Come nel caso del modo 2, anche per il modo grafico 1 è possibile sfruttare i dati elaborati in precedenza con il seguente programma

```
CLS
X$ = CHR$(&H50) + CHR$(&H50) + CHR$(&HFA) + CHR$(&HFA)
SCREEN 1
CIRCLE (100, 100), 100
PAINT (100, 100), X$
END
```

Per provare tutti i motivi, inserire i dati esadecimali per formare la stringa di riempimento X$.

Nei modi grafici usati con le schede EGA e VGA, le cose si fanno un po' più complesse dato che è possibile utilizzare più colori per un pixel. Nel modo 8 (come nel 9 e nel 12), in cui è possibile rappresentare 16 colori per pixel, il calcolo dei valori da usare per la stringa di riempimento è il seguente: si supponga di volere usare il seguente motivo

```
...xx...
...xx...
xxxxxxxx
x......x
x......x
xxxxxxxx
```

che, a schermo pieno, genera una figura tipo 'muro di mattoni', e si supponga inoltre di usare il colore magenta per i pixel evidenziati con un '.' ed il colore giallo per quello con la 'x' (tenere presente che è possibile usare altri 14 colori oltre quelli dell'esempio, anche contemporaneamente); dato che i codici per i colori prescelti sono 0101 (magenta) e 1110 (giallo), scrivere i suddetti codici per ogni pixel della prima riga, dal basso verso l'alto e considerare i valori ottenuti da sinistra verso destra e dall'alto verso il basso, nel seguente modo

```
                calcolo byte
                -------->
                ^ 11100111  →  esadecimale E7 |
verso           | 00011000  →  esadecimale 18 |
per il codice   | 11111111  →  esadecimale FF |
colore          | 00011000  →  esadecimale 18 |
                MMMGGMMM                      V
                   colori               stringa ottenuta
                                         dai caratteri
                                          E8 18 FF 18
```

e così per la terza riga

```
                calcolo byte
                -------->
                ^ 00000000  →  esadecimale 00 |
verso           | 11111111  →  esadecimale FF |
per il codice   | 11111111  →  esadecimale FF |
colore          | 11111111  →  esadecimale FF |
                GGGGGGGG                      V
                   colori               stringa ottenuta
                                         dai caratteri
                                          00 FF FF FF
```

e così per la quarta

```
                calcolo byte
                -------->
                ^ 01111110  →  esadecimale 7E |
verso           | 10000001  →  esadecimale 81 |
per il codice   | 11111111  →  esadecimale FF |
colore          | 10000001  →  esadecimale 81 |
                GMMMMMMG                      V
                   colori               stringa ottenuta
                                         dai caratteri
                                          7E 81 FF 81
```

Considerando inoltre che sono eguali la prima e la seconda riga, la terza e sesta riga, la quarta e quinta riga, la sequenza di bytes da inserire nella stringa risulta la seguente

```
E7 18 FF 18
E7 18 FF 18
00 FF FF FF
7E 81 FF 81
7E 81 FF 81
00 FF FF FF
```

e tutto ciò è riassunto nel seguente esempio che permette di visualizzare a caso dei cerchi riempiti con tale motivo

```
RANDOMIZE TIMER
MT$=CHR$(&HE7) + CHR$(&H18) + CHR$(&HFF) + CHR$(&H18)
MT$=MT$+CHR$(&HE7) + CHR$(&H18) + CHR$(&HFF) + CHR$(&H18)
MT$=MT$+CHR$(&H00) + CHR$(&HFF) + CHR$(&HFF) + CHR$(&HFF)
MT$=MT$+CHR$(&H7E) + CHR$(&H81) + CHR$(&HFF) + CHR$(&H81)
MT$=MT$+CHR$(&H7E) + CHR$(&H81) + CHR$(&HFF) + CHR$(&H81)
MT$=MT$+CHR$(&H00) + CHR$(&HFF) + CHR$(&HFF) + CHR$(&HFF)
SCREEN 12
DO
    X=RND*640
    Y=RND*480
    R=RND*30
    CIRCLE (X, Y), R, 2
    PAINT (X, Y), MT$, 2
    IF RND>.99 THEN CLS
LOOP
END
```

L'opzione 'sfondo' infine, è usata con l'istruzione PAINT nel caso in cui sia necessario riempire con un motivo una figura già colorata in precedenza con un colore in tinta unica; infatti, l'algoritmo usato dall'istruzione PAINT per riempire una figura chiusa, prevede che la colorazione stessa debba terminare quando l'ultimo punto colorato sia attorniato tutto da punti colorati con lo stesso colore; per evitare quindi che il riempimento successivo ad una colorazione non funzioni per problemi legati a tale fatto, viene prevista una stringa come ultimo argomento ('sfondo') che, secondo le regole viste in precedenza, determina la colorazione di una riga che l'algoritmo **non deve considerare** e che quindi non limita il riempimento in corso. Nel seguente esempio, un cerchio colorato in magenta viene riempito con un motivo a strisce orizzontali in 4 colori, specificando che le strisce in magenta **non devono** costituire degli ostacoli al riempimento ma che l'area da riempire deve essere solo delimitata dal colore ciano (codice 1)

```
SCREEN 1
X$=CHR$(&H00) + CHR$(&HAA) + CHR$(&HFF) + CHR$(&H55)
CIRCLE (150, 100), 100, 1
PAINT (150, 100), 2, 1
A$=INPUT$(1)
PAINT (150, 100), X$, 1, CHR$(&HAA)
A$=INPUT$(1)
CLS
END
```

L'ultima istruzione PAINT usata, specifica che l'area da riempire con il motivo contenuto nella stringa X$, è delimitata dal colore ciano (codice 1) e non devono essere considerate delle righe di colore magenta (codice esadecimale AA) in essa contenuta.

*** PALETTE**

Questa istruzione, come la PALETTE USING, è usata solo su sistemi dotati di scheda EGA, VGA o MCGA. Essa serve a cambiare gli assegnamenti standard degli attributi ai relativi colori.

Infatti, anche se le schede in questione permettono di usare moltissimi colori, essi possono essere sempre presenti sul video solo in maniera limitata; è possibile cioè, scegliere un sottoinsieme tra i colori disponibili, sottoinsieme che è contemporaneamente visibile sullo schermo ed a cui si dà il nome di **palette**.

Per cambiare palette è sufficiente modificare il valore del colore assegnato ad un attributo, con l'istruzione di cui sopra che ha la seguente sintassi

PALETTE [**attributo, colore**]

All'avvio del sistema, esiste una palette di default per ogni modalità video, che assegna determinati colori a determinati attributi. Questa palette è sempre ottenibile se si usa l'istruzione senza argomenti, e cioè

PALETTE

Nella modalità 12, ad esempio, il colore verde è assegnato all'attributo 2; se lo si volesse cambiare con il colore giallo, basterebbe eseguire l'istruzione

PALETTE 2, 14

dato che il colore giallo è il 14; così facendo tutti i punti verdi già presenti sul video, diventerebbero automaticamente gialli, e l'uso dell'attributo 2 genererebbe punti gialli.

La palette di default per i vari modi video, è specificata nelle tabelle seguenti

<u>Modi 1 e 9</u>

Attributo	Numero Colore	Colore
0	0	Nero
1	11	Ciano chiaro
2	13	Magenta chiaro
3	15	Bianco intenso

<u>Modi 2 e 11</u>

Attributo	Numero Colore	Colore
0	0	Nero
1	15	Bianco intenso

<u>Modi 0, 7, 8, 9, 12 e 13</u>

Attributo	Numero Colore	Colore
0	0	Nero
1	1	Blu

2	2	Verde
3	3	Ciano
4	4	Rosso
5	5	Magenta
6	6	Marrone
7	7	Bianco
8	8	Grigio
9	9	Blu chiaro
10	10	Verde chiaro
11	11	Ciano chiaro
12	12	Rosso chiaro
13	13	Magenta chiaro
14	14	Giallo
15	15	Bianco intenso

Per modificare l'intera palette in un solo colpo è possibile usare l'istruzione **PALETTE USING** che ha la seguente sintassi

PALETTE USING vettore [**(indice)**]

in cui il primo argomento è un vettore numerico intero (o intero lungo per i modi 11, 12 e 13), i cui elementi indicano i colori assegnati all'intera palette. Così, se ad esempio si volesse ottenere la seguente palette

Attributo	Numero Colore
0	14
1	8
2	4
3	11
4	9
5	0
6	3
7	12
8	15
9	6
10	13
11	2
12	5
13	1
14	10
15	7

basterebbero le seguenti linee di programma

```
DEFINT A-Z
DIM NEWPAL&(0 TO 15)
SCREEN 12
FOR C=0 TO 15
```

```
        READ NEWPAL&(C)
    NEXT C
    PALETTE USING NEWPAL&(0)
    END
    DATA 14,8,4,11,9,0,3,12,15,6,13,2,5,1,10,7
```

Il seguente esempio, inoltre, mostra gli effetti dell'istruzione PALETTE alle immagini già visualizzate

```
RANDOMIZE TIMER
DEFINT A-Z
CLS
SCREEN 13
FOR X=1 TO 100
    LINE (RND*300, RND*200)-(RND*300, RND*200), RND*255
NEXT X
DO
    A=RND*255
    C&=RND*63
    LOCATE 1,1
    PRINT A, C&
    PALETTE A, C&
LOOP WHILE INKEY$<>CHR$(27)
PALETTE
END
```

*** PALETTE USING**

Vedere PALETTE

*** PCOPY**

Tramite questa istruzione è possibile copiare una pagina video in un'altra, se il sistema prevede la gestione di più pagine. La semplice sintassi della istruzione PCOPY, è

PCOPY pagsorg, pagdest

in cui il primo parametro è il numero della pagina da copiare ed il secondo è il numero della pagina in cui deve essere copiata la prima. Il numero massimo che è possibile specificare come numero di pagina, dipende dalla modalità di visualizzazione (imposta dall'istruzione SCREEN) ed anche dalla disponibilità di memoria RAM sulla scheda video utilizzata. In generale, la seguente tabella mostra il numero di pagine disponibili a seconda del modo video prescelto

se si dispone di una scheda **MDA**

Modo	Numero pagine	Risoluzione
0	1	720 x 350

se si dispone di una scheda **CGA**

Modo	Numero pagine	Risoluzione

0	8	320 x 200
0	4	640 x 200
1	1	320 x 200
2	1	640 x 200

se si dispone di una scheda **EGA**

Modo	Numero pagine	Risoluzione	Colori
0	8	320 x 200	16 tra 16
0	8	320 x 350	16 tra 64
0	8	320 x 350	16 tra 64
0	8	640 x 200	16 tra 16
0	8	640 x 350	16 tra 64
0	8	640 x 200	16 tra 16
0	8	720 x 350	16 tra 3
0	4	640 x 350	16 tra 64
0	4	720 x 350	16 tra 3
1	1	320 x 200	4 tra 16
2	1	640 x 200	2 tra 16
7	(a)	320 x 200	16 tra 16
8	(a)	640 x 200	16 tra 16
9 (b)	1	640 x 350	4 tra 64
9 (b)	1	640 x 350	4 tra 64
9 (b)	(a)	640 x 350	16 tra 64
9 (b)	(a)	640 x 350	16 tra 64
10	(d)	640 x 350	4 tra 9
10	(d)	640 x 350	4 tra 9

Note per la EGA:

(a) il numero di pagine è ottenibile dividendo la quantità di memoria disponibile per 32K (modo 7), 64K (modo 8) e 128K (modo 9). Il massimo è 8, il minimo 1;

(b) questi modi sono previsti per una EGA con 64K di memoria video;

(c) questi modi sono previsti per una EGA con più di 64K di memoria video;

(d) il numero di pagine è ottenuto dividendo la quantità di memoria video per due e per 64K. Il massimo è 8, il minimo è 1;

se si dispone di una scheda **VGA**

Modo	Numero pagine	Risoluzione	Colori
0	8	360 x 400	16 tra 64
0	8	320 x 350	16 tra 64
0	4	320 x 400	16 tra 64
0	8	720 x 400	16 tra 64
0	4	640 x 350	16 tra 64
0	4	720 x 350	16 tra 3
0	4	640 x 400	16 tra 64
0	4	720 x 400	16 tra 3
1	1	320 x 200	4 tra 16
2	1	640 x 200	2 tra 16

7	(a)	320 x 200	16 tra 64
8	(a)	640 x 200	16 tra 16
9	(a)	640 x 350	16 tra 64
9	(a)	640 x 350	16 tra 64
10	(b)	640 x 350	4 tra 9
10	(b)	640 x 350	4 tra 9
11	1	640 x 480	2 tra 256K
11	1	640 x 480	2 tra 256K
12	1	640 x 480	16 tra 256K
12	1	640 x 480	16 tra 256K
13	1	320 x 200	256 tra 256K

Note per la VGA:

(a) il numero di pagine è ottenibile dividendo la quantità di memoria disponibile per 32K (modo 7), 64K (modo 8) e 128K (modo 9). Il massimo è 8, il minimo 1;

(b) il numero di pagine è ottenuto dividendo la quantità di memoria video per due e per 64K. Il massimo è 8, il minimo è 1;

se si dispone di una scheda **MCGA**

Modo	Numero pagine	Risoluzione	Colori
0	8	320 x 400	16 attributi
0	8	640 x 400	16 attributi
1	1	320 x 200	4 attributi
2	1	640 x 200	2 attributi
11	1	640 x 480	2 tra 256K
11	1	640 x 480	2 tra 256K
13	1	320 x 200	256 tra 256K

Le tabelle relative alla VGA e MGCA tiene conto solo delle schede con risoluzione massima di 640 x 480; con l’avvento delle VGA 800 x 600 e 1024 x 768, le cose si sono complicate ulteriormente anche perché non sempre sono disponibili i drivers per la loro gestione; date queste premesse, per l’uso di tali schede con QB, riferirsi sempre al loro manuale di riferimento.

Con il presente esempio è possibile mostrare come è possibile sfruttare l’istruzione PCOPY, per ottenere uno scambio veloce di pagine video

```
SCREEN 0, , 0, 0
CLS
PRINT “Questa e’ la pagina 0 … (la principale)”
PRINT “Pressando un tasto, verranno copiate,”
PRINT “alternativamente, in questa la 2 e la 1 …”
SCREEN 0, , 2, 2
CLS
FOR X=1 TO 100
    PRINT “pagina numero 2 …”;
NEXT X
```

```
A$=INPUT$(1)
SCREEN 0, , 1, 1
CLS
FOR X=1 TO 100
      PRINT "PAGINA NUMERO 1 …";
NEXT X
A$=INPUT$(1)
SCREEN 0, , 0, 0
PAG=1
DO
      A$=INPUT$(1)
      PCOPY PAG, 0
      PAG = PAG XOR 3
LOOP WHILE A$<>CHR$(27)
CLS
END
```

Viene richiesta almeno la scheda CGA per questo esempio.

*** PEN**

Vedere ON PEN

*** PLAY (istruzione musicale)**

Questa istruzione, molto potente e versatile, permette di generare suoni molto articolati con il sistema MS-DOS. Essa infatti sfrutta l'altoparlante interno del computer e permette di stabilire durata e frequenza delle singole note prodotte. La sintassi, molto semplice, è la seguente

PLAY stringa

in cui stringa è la stringa di comando il cui contenuto, ottenuto secondo un certo codice, costituisce il motivo da suonare. Per ogni azione, controllo o nota da suonare esiste una o più lettere o simboli da introdurre nella stringa, secondo il seguente codice

On imposta l'ottava. Le ottave sono sette, ed il valore n può andare da 0 a 6
> è il simbolo usato per aumentare l'ottava corrente
< è il simbolo che permette di diminuire l'ottava corrente
A secondo la scala anglosassone, questa lettera rappresenta la nota 'La'
B è un 'Si'
C 'Do'
D 'Re'
E 'Mi'
F 'Fa'
G 'Sol'
Nn serve a suonare la nota n che è compreso tra 0 ed 84 (84 note in 7 ottave). Se n=0 si ottiene una pausa
questo simbolo posto dopo quello di una nota serve ad indicare il fatto che è diesis
+ questo simbolo si comporta come il precedente
- il segno meno serve invece ad ottenere la nota bemolle

Ln serve ad impostare la durata delle note. Il valore specificato va da 1 a 64. È previsto che la durata segua direttamente la nota per impostare la lunghezza solo di quella nota. Ad esempio, B4 indica un Si semiminima

MN imposta il suono normale. Ogni nota dura i 7/8 della durata impostata con L

ML imposta il suono legato. Ogni nota dura esattamente la durata impostata con L

MS imposta il suono staccato. Ogni nota dura i 3/4 della durata impostata con L

Pn imposta una pausa con durata prefissabile da 1 a 64 secondo la stessa scala usata con il comando L (durata delle note)

Tn imposta il tempo (numero di semiminime L 4 al minuto). Il valore predisposto è 120, ma n può variare tra 32 e 255

. un punto dopo una nota ne aumenta la durata di metà dell'aumento impostato dal punto precedente, secondo la serie 1+1/2+1/4+…

MF imposta il suono in primo piano. Ogni nota cioè, non inizia se non termina la nota precedente. Il computer esegue la prossima istruzione dopo la PLAY solo al termine di quest'ultima. È il modo impostato per default

MB l'esecuzione dei suoni è impostata in sottofondo (background). Le note vengono immesse in un buffer ed il programma continua ad essere eseguito anche se queste non sono state tutte suonate. Un sistema che agisce in sottofondo provvede a suonare tutte le note presenti nel buffer (fino a 32)

"X"+VARPTR$(stringa) il comando X insieme all'indirizzo di una stringa contenente codice per generare suoni, viene usato per eseguire tali suoni con l'istruzione PLAY.

Come esempio di uso dell'istruzione PLAY, ecco un piccolo programma che simula la suoneria di un telefono di tipo moderno; vengono simulati 5 squilli di un telefono elettronico

```
FOR T=1 TO 5
    FOR Y=1 TO 5
        PLAY "L16T255ABC"
    NEXT Y
    FOR Y=1 TO 4000
    NEXT Y
NEXT T
```

*** PLAY (controllo gestore eventi)**

Vedere ON PLAY(n)

*** POKE**

È l'istruzione usata per la scrittura diretta nella memoria RAM del sistema. La sua sintassi è

POKE indirizzo, valore

in cui indirizzo è l'offset (da 0 a 65535) riferito al segmento attuale definito con l'istruzione DEF SEG, ed il valore da scrivere è compreso tra 0 e 255 (byte). Ad esempio, se si volesse porre ad 1 il bit numero 6 dell'indirizzo 1047 del segmento 0, locazione prevista dal BIOS per conservare lo stato dei tasti di controllo (Shift, Alt, Ctrl), basterebbero le seguenti istruzioni

```
DEF SEG=0
POKE 1047, PEEK(1047) OR 64
```

DEF SEG

ricordando che la funzione PEEK( ), complementare all'istruzione POKE, serve a leggere dalla memoria RAM il valore contenuto ad un indirizzo specificato. Le istruzioni suddette quindi, prelevano dall'indirizzo 1047 del segmento 0 il valore in esso contenuto, lo modificano settando il bit numero 6 (OR 64) e scrivono il risultato nella stessa locazione di memoria. Così facendo si dovrebbe accendere il led del Caps Lock previsto nella tastiera del PC ed il modo dovrebbe essere il maiuscolo. Attenzione che questo modo di procedere è molto legato all'hardware di sistema e pertanto potrebbe non funzionare su tutti i computers MS-DOS per questioni di compatibilità.

È comunque buona norma usare con molta attenzione l'istruzione POKE perché scritture di valori all'interno di zone di memoria di cui non si conosce bene la funzione, possono far 'cadere' il sistema, cioè bloccare il sistema operativo o i programmi che si stanno eseguendo.

*** PRESET**

È una istruzione grafica usata per visualizzare un pixel sul video in modalità grafica. È analoga a PSET ed adotta la stessa sintassi di quest'ultima

PRESET [**STEP**] **(xc,yc)** [**,colore**]

La clausola STEP viene specificata quando le coordinate del punto specificate tra parentesi si intendono relative all'ultima posizione occupata dal cursore grafico. Quest'ultimo non è visibile ma viene spostato da ogni istruzione grafica alle coordinate dell'ultimo punto utilizzate dall'ultima istruzione. All'inizio il cursore grafico è posto al centro dello schermo.

La differenza tra PRESET e PSET è data dal fatto che, omettendo il colore nella PRESET, il punto viene disegnato con il colore di sfondo (e quindi cancellato), mentre, omettendo il colore nella PSET, viene disegnato il punto con il colore di primo piano (e quindi attivato).

Ecco un esempio che coinvolge sia l'istruzione PSET che la PRESET

```
CONST PI=3.1415, XC=300, YC=100
SCREEN 2
CLS
FOR RAGGIO=1 TO 200 STEP 10
      FOR ANG=0 TO PI*2 STEP PI/20
            PSET (XC+SIN(ANG)*RAGGIO, YC+COS(ANG)*RAGGIO*.5)
      NEXT ANG
NEXT RAGGIO
```

*** PRINT**

È forse l'istruzione più usata nella programmazione in Basic in quanto è quella che permette la visualizzazione dei dati sul video. La sua sintassi è

PRINT [**elencoespres**][{**;** | **,**}]

e si può abbreviare con

? [**elencoespres**][{**;** | **,**}]

Se compare senza argomenti, l'istruzione PRINT visualizza sullo schermo una riga vuota alla posizione corrente del cursore e serve ad annullare eventuali effetti di virgole o punti e virgola posti alla fine di istruzioni PRINT precedenti.

Elencoespres può essere un elenco di espressioni numeriche e/o alfanumeriche separate da una virgola o da un punto e virgola; la differenza nell'uso di questi due simboli è data la posizione di visualizzazione su video. Infatti, nel caso del punto e virgola, i dati (risultati delle espressioni in oggetto), sono visualizzati affiancati l'uno all'altro sulla stessa riga, mentre, con la virgola sono separati da 14 spazi (campo di visualizzazione).

Il punto e virgola o la virgola lasciata alla fine di una istruzione PRINT, sono riconosciuti dalla prossima istruzione PRINT che visualizza i propri dati in accordo con le specifiche del simbolo separatore utilizzato nella precedente PRINT.

I valori numerici sono visualizzati dall'istruzione PRINT facendoli precedere da uno spazio se positivi, da un segno '-' se negativi; in ambedue i casi, comunque, al numero segue sempre uno spazio. I dati alfanumerici vengono invece visualizzati senza alcuna modifica.

Nell'esempio seguente vengono mostrate le differenze nei campi di visualizzazione di alcune PRINT

```
CLS
PRINT “123456789012345678901234567890123456789012345”
PRINT 1, -2, 3, -4
PRINT “A”, “B”, “C”, “D”
PRINT 1; -2; 3; -4
PRINT “A”; “B”; “C”; “D”
```

infatti, questo programma di prova, fornisce il seguente output

```
123456789012345678901234567890123456789012345
 1     -2      3     -4
A      B      C      D
 1 -2 3 -4
ABCD
```

Per produrre un output su carta anziché su video, è sufficiente usare l'istruzione **LPRINT** anziché la PRINT; questa istruzione ha la stessa sintassi delle PRINT e si comporta allo stesso modo, soltanto che dirige l'output verso la stampante collegata. È comunque consigliabile l'uso della funzione **USING**, sia per la PRINT che per la LPRINT, per produrre tabulati esteticamente corretti.

*** PRINT #**

L'istruzione PRINT, normalmente rivolta al monitor, può essere usata per scrivere dati su un'altra periferica con l'aggiunta del simbolo #. Dopo questo simbolo, può essere infatti specificato un numero di file, aperto in precedenza con una istruzione OPEN. Tale file, se

sequenziale, deve essere stato aperto in output o in append. Può anche fare riferimento ad una periferica aperta con una istruzione OPEN (ad esempio, una porta di comunicazione seriale) a condizione che sia una periferica di output.

Per quanto riguarda i file sequenziali, porre attenzione al fatto che l'istruzione PRINT # scrive su file le informazioni nello stesso formato con cui le farebbe apparire sul monitor l'istruzione PRINT. Ad esempio, usando l'istruzione

PRINT #1, 1, 2, 3

sul file sequenziale aperto sul canale 1, verrebbe scritta la sola, seguente riga

 1 2 3

con gli spazi tra i dati a causa delle virgole di separazione e **senza alcun separatore** come la virgola o il carattere 13 ASCII. Ciò comporta un errore al momento in cui, tramite l'istruzione INPUT #, si tenta di leggere i tre valori dal file; infatti, per l'assenza di separatori, questi vengono considerati come un unico dato e si verifica un errore di esecuzione alla seconda istruzione INPUT #.
Per evitare questo inconveniente, si possono scegliere due metodi

a) si predispongono i separatori nell'istruzione PRINT #; l'esempio precedente diventa

PRINT #1, 1, ",",2,",",3

(notare come le virgole siano considerate delle costanti alfanumeriche da scrivere nel file così come introdotte nel programma);

b) si usa una istruzione PRINT # per ogni dato che si intende scrivere su file; ad esempio

PRINT #1, 1
PRINT #1, 2
PRINT #1, 3

(in tale modo vengono introdotte delle sequenze CR-LF come separatori tra i dati).

Al posto dell'istruzione PRINT #, per i file sequenziali, è più opportuno usare l'istruzione WRITE # che rimedia automaticamente a questi piccoli problemi.

*** PRINT USING**

Tramite la funzione **USING**, l'istruzione PRINT acquista potenza e permette di risolvere tutti i problemi di presentazione dei dati su video o su tabulato cartaceo (**LPRINT USING**).

La sintassi dell'istruzione è la seguente

PRINT USING stringaform; espr1[**, …**]

La stringaform è quella stringa che contiene i comandi di formattazione dei dati in uscita. I comandi usati per la formattazione sono i seguenti

! stampa solo il primo carattere di una stringa anche se questa è molto lunga;

**\ ** stampa solo 2+n caratteri di una stringa; n è il numero di spazi che si includono tra i due backslashes. Per stampare i primi due caratteri, non si metta aggiunga spazio (\\). Se la stringa è più lunga del campo, viene troncata a destra; se il campo è più lungo della stringa, vengono aggiunti spazi a destra

& stampa la stringa così com'è. Questo simbolo indica un campo alfanumerico di lunghezza variabile

indica una cifra in un campo numerico. Devono essere specificate tutte le cifre che compongono il numero; questo sarà allineato a destra. Se il numero è più grande di quello esprimibile con la maschera specificata, viene stampato il numero completo preceduto dal simbolo **%**. È possibile sfruttare fino ad un massimo di 24 cifre

. indica il punto decimale all'interno di una maschera numerica. Se posto dopo un simbolo #, permette di stampare la cifra prima del punto anche se questa è zero; si migliora così la visualizzazione dei valori compresi tra 0 e 1

+ questo simbolo, posto all'inizio o alla fine di un campo numerico, permette la visualizzazione del segno del numero. Se il numero è positivo, non viene visualizzato uno spazio ma il segno +, se è negativo il segno –

- il simbolo - può essere posto solo alla fine di un campo numerico per visualizzare il segno del numero. Se positivo, il numero viene seguito da uno spazio altrimenti dal segno -

****** se questo simbolo viene messo all'inizio di un campo numerico, nella visualizzazione del numero questo sarà completato a sinistra da asterischi, anziché da spazi, se non completamente utilizzato. Questo simbolo specifica due cifre in più nel campo numerico

$$ posto all'inizio di un campo numerico permette di visualizzare il simbolo del dollaro immediatamente alla sinistra del valore. Se il valore è negativo, il segno - viene posto prima del dollaro. Questo simbolo specifica due cifre in più nel campo numerico

****$** con questo simbolo si ottiene l'effetto ottenuto con ** e $$. Se posto all'inizio di un campo numerico, fa apparire il dollaro subito a sinistra del valore e riempie gli spazi vuoti alla sinistra dello stesso con asterischi. Il simbolo **$ specifica tre cifre in più nel campo numerico

, posta alla sinistra del punto decimale in un campo numerico (o comunque all'interno del campo), predispone la visualizzazione della virgola ogni tre cifre del numero a partire da destra

^^^^
^^^^^ questi due simboli (il primo per valori con esponente <100, il secondo per valori con esponente >99), servono a visualizzare i numeri in forma esponenziale. Devono seguire un campo numerico di definizione della mantissa; l'esponente del numero viene aggiustato automaticamente in visualizzazione.

Da notare che, se il valore numerico da visualizzare eccede in grandezza il campo predisposto, tale valore viene stampato per intero ma preceduto dal simbolo **%**. Tale simbolo è quindi un avvertimento del fatto che il campo deve essere ingrandito.

Seguono degli esempi esplicativi per chiarire il funzionamento di tutti i codici della stringa di formattazione della PRINT USING

```
PRINT USING "!"; "ROSSI PAOLO"
PRINT USING "\  \"; "ROSSI PAOLO"
PRINT USING "\            \"; "ROSSI PAOLO"
PRINT USING "###"; 15
PRINT USING "###.##"; 213.567
PRINT USING "+###"; 19
PRINT USING "-###"; -23
PRINT USING "###"; 1000
PRINT USING "###.##"; 969
PRINT USING "**###.##"; 90
PRINT USING "$$##"; 12
PRINT USING "**$##"; 12
PRINT USING "#####,.##"; 15656.2
PRINT USING "#.##^^^^"; 2.2E+12
PRINT USING "###^^^^^": 2.2D+120
```

i cui risultati a video, sono i seguenti

R (stampa il primo carattere)
ROSS (stampa i primi 4 caratteri perché il campo è formato da \ 2 spazi e \)
ROSSI PAOLO (stampa la stringa completa perché il campo è più lungo. Vengono stampati anche gli spazi necessari a completare il campo)
15 (stampa il valore preceduto da uno spazio perché il campo è composto da 3 cifre)
213.57 (stampa il numero approssimando i decimali all'ultima cifra decimale specificata nel formato)
+19 (predispone la stampa del simbolo + perché il valore è positivo)
-23 (come nel precedente esempio, ma il valore è negativo)
%1000 (il campo è troppo piccolo per il valore da stampare e questo viene stampato completamente e preceduto dal simbolo di avvertimento %)
969.00 (anche se il valore non ha parte decimale, questi vengono stampati perché specificato nel formato)
*****90.00** (il valore viene stampato con gli asterischi di completamento)
$12 (viene posto il simbolo del dollaro come da formato)
****$12** (gli spazi prima del dollaro sono riempiti da asterischi come da formato)
15,656.20 (la virgola viene posta ogni tre cifre perché specificato nel formato)

0.22E+13 (l'esponente viene aumentato dato il formto della mantissa)
22D+119 (come nel caso precedente per valori con esponente >99)

Con la funzione USING, l'istruzione PRINT può essere usata anche per mandare i dati su periferica. In questo caso la sintassi completa dell'istruzione diventa la seguente

PRINT #canale, USING stringaform; espr1[**, …**]

Questa forma dell'istruzione permette di registrare su disco i dati già formattati per una eventuale stampa in differita di tabulati. Un uso, alquanto elaborato, di tale istruzione può essere fatto per risolvere un piccolo ma fastidioso problema. Infatti, i valori numerici possono solamente essere stampati con una maschera di formattazione di tipo americano, che presuppone l'uso della virgola dove gli europeei mettono il punto e viceversa; ad esempio, il valore 1.334.567,9 (scritto secondo il formato europeo), negli Stati Uniti viene scritto 1,334,567.9, come del resto è fatto dall'istruzione PRINT USING seguente

```
PRINT USING “#######,.#”; 1334567.9
```

che dà appunto, come risultato

1,334,567.9

Per rimediare, si può inviare ad un file il risultato dell'elaborazione per poi rileggerlo in una variabile e modificarlo. Si devono usare degli accorgimenti per usare questo metodo, e cioè

- è consigliabile usare, per motivi di velocità, un drive creato in memoria (RAMDISK) inserendo, ad esempio, nel file CONFIG.SYS, il comando **Device=Ramdrive.sys 16** che predispone una Ram Disk da 16K;

- si deve utilizzare l'istruzione **LINE INPUT #** per rileggere da file il dato scritto, perché in questo sono presenti delle virgole che disturberebbero la lettura fatta tramite INPUT #;

```
CLS
OPEN “O”, 1, “E:TMP”
PRINT #1, USING “#########,.###”; 12345678.55#
CLOSE
OPEN “i”, 1, “E:TMP”
LINE INPUT #1, A$
CLOSE
FOR LL=1 TO LEN(A$)
  AA=ASC(MID$(A$, LL, 1))
  IF AA=46 THEN
          MID$(A$, LL, 1))=”,”
  ELSE
          IF AA=44 THEN
                  MID$(A$, LL, 1))=”.”
          END IF
  END IF
```

```
NEXT LL
KILL "E:TMP"
PRINT A$
```

In questo esempio si presuppone che il disco virtuale sia stato creato e che sia E:.
Naturalmente, servendosi di questo esempio, è necessari articolare una funzione più generica che trasformi un dato formattato con il metodo americano in quello europeo.

*** PSET**

Vedere PRESET

*** PUT**
(Istruzione di I/O)

La sintassi dell'istruzione PUT, usata per l'I/O dei file random, è la seguente

PUT [**#**]**numfile**[**,**[{**numrecord** | **byte**}]**, variabile**]

in cui 'numfile' è il numero di file associato all'istruzione OPEN corrispondente. Nel caso dei file random, il secondo parametro, se presente, è il numero del record da scrivere; invece, nel caso dei file binari, è la posizione all'interno dello stesso da cui cominciare a scrivere.
Il numero del record, per i file random, può essere compreso tra 1 e 2147483647 mentre la posizione che è possibile specificare, può essere compresa nei limiti della lunghezza del file aperto. Se questo parametro manca, i valori assunti sono quelli correnti; nel caso del file random, all'apertura il numero di record iniziale è 1 e poi, in seguito a letture o scritture, questo viene aggiornato; lo stesso avviene nel caso di file binari per quanto riguarda la posizione di scrittura.
Nel caso di un file binario, il parametro posto alla fine dell'istruzione è rappresentato da una variabile stringa che rappresenta il buffer di scrittura e la sua lunghezza, dichiarata in precedenza, rappresenta il numero di bytes da scrivere; per un file random, essa è invece una variabile di tipo definito dall'utente in quanto rappresenta il record usato dal file stesso; in questo caso non si deve usare l'istruzione FIELD all'apertura del file random (come per l'istruzione GET).
Il seguente esempio di scrittura su file random è complementare a quello usato nella descrizione dell'istruzione GET; esso serve a preparare il file di dati su cui agirà in lettura il programma di esempio dell'istruzione GET

```
CONST TRUE=1
OPEN "R",#1,"PROVA",32
FIELD #1, 15 AS COGNOME$, 15 AS NOME$, 2 AS ETA$
DO WHILE TRUE
    READ C$, N$, E%
    IF C$="*" THEN
        EXIT
    END IF
    LSET COGNOME$=C$
    LSET NOME$=N$
    LSET ETA$=MKI$(E%)
    PUT #1
```

```
LOOP
CLOSE #1
END
DATA ROSSI, PAOLO, 34
DATA VERDI, ENRICO, 21
DATA GIULIANA, ANTONIO, 23
DATA CAIROLI, ANNA, 22
DATA UGHETTI, UGO, 26
DATA ROSI, PIERO, 33
DATA *, *, 0
```

La scrittura avviene in modo sequenziale (attenzione, con **accesso** sequenziale, anche se il file è di tipo random), dato che non è specificato il numero di record dopo l'istruzione PUT.

Come nel caso della GET, si può evitare l'uso della FIELD e della funzione CVI, se si usa la definizione del record con l'istruzione TYPE..END TYPE

```
TYPE RECPROVA
        COGNOME AS STRING * 15
        NOME AS STRING * 15
        ETA AS INTEGER
END TYPE
DIM PREC AS RECPROVA
OPEN "R",#1,"PROVA",LEN(PREC)
        …
        PREC.COGNOME=C$
        PREC.NOME=N$
        PREC.ETA=E%
        PUT #1,,PREC
        …
DATA *, *, 0
```

Notare che nell'istruzione PUT compaiono due virgole per indicare che il numero di record non è specificato, e che le istruzioni LSET sono sostituite da assegnazioni dirette ai componenti del record (come nell'analogo caso per la GET).

*** RANDOM**

Vedere OPEN

*** RANDOMIZE**

Con questa istruzione, posta generalmente all'inizio dei programmi, viene inizializzato quel meccanismo interno del Quick Basic che permette di generare dei numeri casuali tramite la funzione RND. Infatti, la generazione dei numeri casuali si basa di una formula per mezzo della quale si sceglie una sequenza di numeri molto lunga, disposti a caso, dalla quale si scelgono i valori in sequenza. Se l'istruzione RANDOMIZE non fosse usata, la generazione dei numeri casuali partirebbe sempre dalla stessa sequenza e quindi i valori scelti sarebbero, nell'ordine, sempre gli stessi. Per scegliere la sequenza, è necessario specificare un numero generatore (**seme**) compreso tra -32768 e 32767; la sintassi dell'istruzione è la seguente

RANDOMIZE [**seme**]

Se il valore non viene specificato (dato che, nella sintassi, è opzionale), esso viene richiesto automaticamente all'esecuzione dell'istruzione; se, invece, viene usato un valore insieme all'istruzione, questo è usato per scegliere la sequenza di numeri casuali; attenzione al fatto che, se il numero che viene indicato è sempre lo stesso, sempre uguale sarà la sequenza scelta e di conseguenza i numeri casuali. Per ovviare a questo inconveniente, evitando di far richiedere il valore all'inizio del programma, è possibile usare la funzione **TIMER** che ritorna il valore del tempo continuamente aggiornato dal QB in base al clock interno. Usando tale funzione, il numero seguente l'istruzione RANDOMIZE cambierà da esecuzione ad esecuzione del programma, e quindi anche la sequenza di numeri casuali sarà diversa.

Come esempio di uso di RANDOMIZE, si veda il seguente programma che sceglie, tra un mazzo di 52 carte, un gruppo di 5 carte con i relativi semi; l'uso dell'array CARTE% è necessario per evitare che nel gruppo di 5 carte possano esserci delle carte ripetute

```
RANDOMIZE TIMER
DIM CARTE%(1 TO 13, 1 TO 4)
CLS
RISP$="S"
DO WHILE NOT UCASE$(RISP$)="N"
        FOR C=1 TO 13
                FOR S=1 TO 4
                        CARTE%(C, S)=0
                NEXT S
        NEXT C
        PRINT "--- CARTE SELEZIONATE ---"
        FOR C=1 TO 5
                DO
                        CARTA=INT(RND*13+1)
                        SEME=INT(RND*4+1)
                LOOP WHILE CARTE%(CARTA, SEME)=1
                CARTE%(CARTA, SEME)=1
                IF CARTA>10 OR CARTA=1 THEN
                        SELECT CASE CARTA
                                CASE 1
                                        PRINT " A";
                                CASE 11
                                        PRINT " J";
                                CASE 12
                                        PRINT " Q";
                                CASE 13
                                        PRINT " K";
                  END SELECT
          ELSE
                PRINT USING "##"; CARTA;
          END IF
```

```
        PRINT CHR$(2+SEME); " ";
    NEXT C
    PRINT
    PRINT
    INPUT "ANCORA ? "; RISP$
    LOOP
    END
```

*** READ**

È l'istruzione che, insieme a DATA e RESTORE, permette di assegnare dei valori a delle variabili direttamente da programma. L'istruzione READ legge dalle frasi DATA presenti nel modulo principale (e solo in questo) i dati, e li assegna a determinate variabili. La sintassi dell'istruzione è

READ var1[**,var2, …**]

in cui var1, var2, … sono i nomi delle variabili, semplici o con indice, di qualsiasi tipo, in cui si vogliono depositare i dati delle frasi DATA.
Ogni elemento specificato nelle frasi DATA viene copiato all'interno della variabile specificata al momento dell'esecuzione della READ. Se viene tentata una lettura di dati in più rispetto agli elementi presenti nelle frasi DATA, si ha un errore; gli elementi non letti, invece, verranno ignorati.

Se viene eseguita una istruzione RESTORE, le istruzioni READ seguenti leggeranno i dati dalla frase DATA indicata nell'istruzione RESTORE stessa o, in mancanza di specifiche, dalla prima frase DATA.

Viene inoltre segnalato un errore nella istruzione READ che utilizza una variabile numerica, se il corrispondente elemento da leggere nelle frasi DATA è di tipo alfanumerico; se, invece, la variabile utilizzata è alfanumerica e il tipo di dato da leggere è numerico, questo viene convertito automaticamente in stringa (come con la funzione STR$).

Ecco come l'esempio dell'istruzione precedente potrebbe essere migliorato con l'uso delle frasi DATA e dell'istruzione READ

```
RANDOMIZE TIMER
DIM CARTE%(1 TO 13, 1 TO 4)
DIM VALORI$(1 TO 13)
DIM SEMI%(1 TO 4)
FOR V=1 TO 13
    READ VALORI$(V)
NEXT V
FOR S=1 TO 4
    READ SEMI%(S)
NEXT S
CLS
RISP$="S"
DO WHILE NOT UCASE$(RISP$)="N"
```

```
        FOR C=1 TO 13
            FOR S=1 TO 4
                CARTE%(C, S)=0
            NEXT S
        NEXT C
        PRINT "--- CARTE SELEZIONATE ---"
        FOR C=1 TO 5
            DO
                CARTA=INT(RND*13+1)
                SEME=INT(RND*4+1)
            LOOP WHILE CARTE%(CARTA, SEME)=1
            CARTE%(CARTA, SEME)=1
            PRINT VALORI$(CARTA); CHR$(SEMI%(SEME)); "  ";
        NEXT C
        PRINT
        PRINT
        INPUT "ANCORA ? "; RISP$
    LOOP
    END
    DATA A,2,3,4,5,6,7,8,9,10,J,Q,K
    DATA 3,4,5,6
```

*** REDIM**

Insieme all'istruzione DIM, permette di allocare memoria per gli arrays. La sintassi di questa istruzione, del tutto simile a quella della DIM, è la seguente

REDIM var1([**indinf1 TO**] **indsup1)**[**,var2(**[**indinf2 TO**] **indsup2),…**]

Questa istruzione serve a modificare la quantità di memoria predisposta per gli arrays a condizione che questi siano di tipo dinamico. Infatti, per gli arrays dimensionati e dichiarati di tipo statico, l'istruzione REDIM fa emettere al QB un errore. Se non altrimenti specificato, gli arrays usati con REDIM sono intesi in modo dinamico.
Non è comunque possibile variare il numero degli indici degli arrays sia che quest'ultimi siano statici che dinamici.

Ecco alcuni esempi di uso di REDIM corretti

```
' (Esempio 1)
' $DYNAMIC
DIM A$(1000)
…
REDIM A$(150)

' (Esempio 2)
' $DYNAMIC
REDIM A$(150)
…
REDIM A$(200)
```

mentre questi sono errati

```
' (Esempio 3)
' $STATIC
DIM A$(120)
…
REDIM A$(1000)

' (Esempio 4)
' $DYNAMIC
DIM A$(1000)
…
REDIM A$(10, 10)
```

*** REM**

Questa istruzione, molto utile ma, purtroppo, troppo poco sfruttata, serve a porre dei commenti all'interno delle righe di un programma.
Dopo ogni istruzione REM, può essere posta qualsiasi frase termini prima della fine della riga stessa.
Se il commento è posto, in una riga, dopo un comando valido, questi devono essere separati dal simbolo ":".

Questa istruzione può essere specificata anche con una singola virgoletta (') ed in questo caso può anche non essere separata dal simbolo ":" se non è sola su una riga. Se un commento viene usato dopo una frase DATA, il simbolo di separazione (":") è obbligatorio per evitare malfunzionamenti.

Ad esempio

```
' (Main Module)
FOR T=1 TO 5        : REM Uso corretto
        READ T      ' Anche questo uso è corretto
        PRINT T
NEXT T
READ A$
PRINT A$
END
DATA 1,5,6,8,9      ' Uso scorretto (manca il :)
DATA ROSSI PAOLO
```

Porre infine attenzione all'uso dei **metacomandi** che, pur essendo delle parole poste dopo una istruzione di nota, costituiscono a tutti gli effetti delle istruzioni (o meglio direttive) che vengono eseguite. Questi sono **$STATIC**, **$DYNAMIC** e **$INCLUDE**.

*** RESET**

Questa istruzione, poco conosciuta ed altrettanto poco usata del QB serve solamente a chiudere **tutti** i file aperti ed a, eventualmente, svuotarne il buffer. La sua sintassi non prevede alcun argomento; questa istruzione è completamente sostituibile con la CLOSE.

*** RESTORE**

La sintassi di questa istruzione è la seguente

RESTORE [{**numriga** | **etichetta**}]

Essa serve a posizionare il puntatore alla prossima frase DATA da leggere con l'istruzione READ. È infatti possibile rileggere delle frasi DATA già lette e, prima di farlo, bisogna usare l'istruzione RESTORE con, eventualmente il riferimento alla linea dove è presente la frase DATA da rileggere. Questo riferimento può essere costituito da un numero di linea o da una etichetta; queste possono trovarsi anche prima della frase DATA che sarà comunque letta. Se tale riferimento dovesse mancare, verrà riposizionato il puntatore alla **prima** frase DATA presente a livello del modulo.

Ad esempio,

```
CLS
PRINT "PRIMA LETTURA"
FOR T=1 TO 5
        READ X
        PRINT X;
NEXT T
RESTORE VALORI
PRINT "SECONDA LETTURA"
FOR T=1 TO 5
        READ X
        PRINT X;
NEXT T
END
VALORI:
DATA 2,3,22,33,1
```

*** RESUME**

Vedere ON ERROR GOTO

*** RETURN**

Vedere GOSUB

*** RMDIR**

Come per il comando del DOS, questa istruzione serve a rimuovere una sottodirectory esistente all'interno di un disco. La semplice sintassi

RMDIR directory

indica che il comando deve essere seguito dal nome della directory da cancellare; come in DOS con il comando equivalente, la directory può essere cancellata solo se è vuota; il comando RMDIR non può essere abbreviato in RD come in DOS.

Ad esempio, questo programma elimina una directory a richiesta dell'utente

```
ON ERROR GOTO ERTRAP
PRINT
PRINT "REMOVE DIRECTORY – VER. 1.1"
CM$=COMMAND$
IF CM$="" THEN
        PRINT "USO : RMD DIRNAME"
        BEEP
        END
END IF
RMDIR CM$
PRINT "CM$; " RIMOSSA."
END
ERTRAP:
IF ERR=76 THEN
        PRINT "ERRORE DURANTE LA RIMOZIONE DI "; CM$
        END
END IF
END
```

Il programma, chiamato RMD.BAS, compilato con il comando

BC /E RMD;

e linkato con

LINK RMD;

permette di rimuovere la directory specificata sulla linea di comando al pari del comando interno del DOS.

*** RSET**

Vedere LSET

*** RUN**

Il comando RUN, usato per avviare l'esecuzione di un programma, non è molto sfruttato in Quick Basic; la sua sintassi è la seguente

RUN [**numlinea** | **filepec**]

in cui numlinea è un numero di linea valido all'interno di un programma da cui iniziare l'esecuzione dello stesso; se viene usato il comando RUN senza numero di linea, l'esecuzione

parte dalla prima. Se il numero di linea non c'è o si specifica una etichetta alfanumerica, viene segnalato un errore.

Si può avviare un programma dopo averlo caricato automaticamente, specificandone il nome dopo il comando RUN; tale nome ha, per default, estensione .BAS se si ci trova all'interno dell'ambiente QB, mentre è .EXE se si è in DOS. Ad esempio, il seguente programma

```
CLS
FOR T=1 TO 100
        PRINT T
NEXT T
RUN "PARTE2"
```

dopo aver eseguito il ciclo FOR..NEXT, richiama il programma PARTE2 (PARTE2.BAS se si è in ambiente QB o PARTE2.EXE se si è in DOS) e lo esegue; **prestare attenzione** al fatto che tutti i file aperti dal programma chiamante **vengono chiusi** all'esecuzione dell'istruzione RUN tramite la quale, in Quick Basic, non è possibile farli rimanere aperti; per problemi del genere si usa l'istruzione **CHAIN**.

L'istruzione RUN si può usare anche in modo Immediate, ma viene sostituita in questo caso, dall'uso dei tasti Shift F5-

È possibile usare il comando RUN richiamando solo del codice presente a livello di modulo e non di Sub o Function.

*** SCREEN**

È l'istruzione usata per predisporre la modalità di visualizzazione, in dipendenza della scheda video e del monitor a disposizione. La sua sintassi è

SCREEN [**modo**][**,**[**flagcolore**]][**,**[**pagout**]][**,**[**pagvis**]]

Il parametro 'modo' indica la modalità video da usare, tra le possibili nella seguente tabella

Modo 0	- solo modo testo - il formato può essere di 40 x 25, 40 x 43, 40 x 50, 80 x 25, 80 x 43, 80 x 50 caratteri a seconda della scheda video disponibile - si possono usare 2 tra 16 colori; se la scheda disponibile è una EGA, essi sono 16 tra 16
Modo 1	- media risoluzione grafica - sono previsti 320 x 200 pixels (40 x 25 caratteri) - si possono usare due gruppi di 4 tra 16 colori - questa modalità è supportata dalla scheda CGA, dalla EGA e dalla VGA
Modo 2	- alta risoluzione grafica - sono previsti 640 x 200 pixels (80 x 25 caratteri) - si possono usare solo il bianco e il nero - questa modalità è supportata dalla scheda CGA, dalla EGA e dalla VGA

Modo 3	- da usarsi solo dopo aver caricato il driver QBHERC.COM (QB 4.0.) o MSHERC.COM (QB 4.5.) - alta risoluzione grafica - sono previsti 720 x 348 pixels (80 x 25 caratteri) - si possono usare solo il bianco e il nero - questa modalità è supportata soltanto dalla scheda Hercules
Modo 7	- media risoluzione grafica - sono previsti 320 x 200 pixels (40 x 25 caratteri) - si possono usare 16 colori - questa modalità è supportata dalla scheda EGA e VGA
Modo 8	- alta risoluzione grafica - sono previsti 640 x 200 pixels (80 x 25 caratteri) - si possono usare 16 colori - questa modalità è supportata dalla scheda EGA e VGA
Modo 9	- alta risoluzione grafica - sono previsti 640 x 350 pixels (80 x 25 o 80 x 43 caratteri) - si possono usare 16 colori tra 64 se la scheda ha a disposizione più di 64K di memoria, altrimenti 4 colori tra 16 - questa modalità è supportata dalla scheda EGA e VGA
Modo 10	- alta risoluzione grafica - sono previsti 640 x 350 pixels (80 x 25 o 80 x 43 caratteri) - si possono usare 4 tra 9 pseudocolori - questa modalità è supportata dalla scheda EGA e VGA
Modo 11	- altissima risoluzione grafica - sono previsti 640 x 480 pixels (80 x 30 o 80 x 60 caratteri) - si possono usare 2 colori tra 256K - questa modalità è supportata dalla scheda VGA e MCGA
Modo 12	- altissima risoluzione grafica - sono previsti 640 x 480 pixels (80 x 30 o 80 x 60 caratteri) - si possono usare 16 colori tra 256K - questa modalità è supportata dalla scheda VGA
Modo 13	- media risoluzione grafica - sono previsti 320 x 200 pixels (40 x 25 caratteri) - si possono usare 256 colori tra 256K - questa modalità è supportata dalla scheda VGA e MCGA

È possibile, a seconda del modo prescelto, del tipo di scheda video a disposizione e della sua dotazione di memoria RAM, avere a disposizione più pagine video (1, 4 o 8); il numero della pagina prescelta per la visualizzazione in un determinato momento è stabilito dall'ultimo parametro dell'istruzione SCREEN (se non specificato, la pagina visualizzata è sempre la prima);

si può specificare, con il parametro ‘pagout’, il numero della pagina su cui indirizzare l’output delle istruzioni grafiche e di visualizzazione dei dati, e questa può essere diversa da ‘pagvis’; questa caratteristica consente di effettuare delle piccole animazioni perché permette di preparare un’immagine senza disturbare la visione della precedente, e così via come nei fotogrammi delle pellicole cinematografiche; questa tecnica comunque, deve essere supportata da programmi molto veloci ed efficienti scritti, anche, con linguaggio misto (QB, C ed Assembler).

Il parametro ‘flagcolore’, se diverso da zero, indica che possono essere usati i colori se la scheda e la modalità prescelta lo consentono; questo parametro va usato solo con i modi video minori di 2.

Per determinare il tipo di hardware video disponibile in un sistema, può essere utile il programma di esempio, incluso nella librerira BPLUS, chiamato GETDCC (Display Combination Code).

Per controllare le modalità supportate dal sistema in uso, è utile usare il seguente programma di test, che funziona correttamente su sistemi dotati di schede video avanzate (EGA, VGA)

```
' (Main Module Test.Bas)
' $INCLUDE: 'BPLUS.INC'
' $INCLUDE: 'DOS.INC'
ON ERROR GOTO GRERR
CLS
DIM X%(1 TO 32)
FOR SC=0 TO 13
CLS
1:
SCREEN SC
PRINT "MODO "; SC
REGX.AX=&H1B00
REGX.BX=0
REGX.DI=VARPTR(X%(1))
REGX.ES=VARSEG(X%(1))
INTERRUPTX &H10, REGX, REGX
DEF SEG=REGX.ES
MAXCOLU = PEEK(REGX.DI+&H6)*256+PEEK(REGX.DI+&H5)
MAXROWS = PEEK(REGX.DI+&H22)
MAXCOLS = PEEK(REGX.DI+&H28)*256+PEEK(REGX.DI+&H27)
MAXPAGS = PEEK(REGX.DI+&H29)
MAXVRAM = (PEEK(REGX.DI+&H31)+1)*64
DEF SEG
R=1
2:
        WINDOW SCREEN (0,0)-(1,1)
        MAXX = PMAP(1, 0)+1
        MAXY = PMAP(1, 1)+1
        WINDOW
```

```
ERWIN:
        PRINT "Max. Num. Colonne : "; MAXCOLU
        PRINT "Max. Num. Righe : "; MAXROWS
        PRINT "Numero Colori : "; MAXCOLS
        PRINT "Numero Pagine : "; MAXPAGS
        PRINT "Risoluzione : ";
        IF R=0 THEN
                PRINT " Solo Modo Testo"
        ELSE
                PRINT MAXX; "x"; MAXY
        END IF
        PRINT "Memoria Video (K) : "; MAXVRAM
FRERR:
        A$=INPUT$(1)
NEXT SC
END
GRERR:
        IF ERL=2 THEN
                R=0
                RESUME ERWIN
        END IF
        PRINT "MODO : "; SC
        PRINT
        PRINT "Modo non supportato."
        RESUME FRERR
```

Notare che i due file inclusi contengono la dichiarazione della INTERRUPTX e della variabile REGX di tipo adeguato all'uso con la stessa INTERRUPTX. Il programma può essere compilato sotto DOS con i comandi

```
BC /E /X TEST;
LINK TEST,,NUL,BPLUS;
```

o eseguito nell'ambiente a condizione di caricare la libreria BPLUS.QLB all'inizio della sessione con il comando

```
QB TEST /LBPLUS
```

L'output del programma, sul video del mio sistema, è stato il seguente

Modo 0
Max. Num. Colonne : 80
Max. Num. Righe : 25
Numero Colori : 16
Numero Pagine : 8
Risoluzione : Solo Modo Testo
Memoria Video (K) : 256

Modo 1
Max. Num. Colonne : 40
Max. Num. Righe : 25
Numero Colori : 4
Numero Pagine : 1
Risoluzione : 320 x 200
Memoria Video (K) : 256

Modo 2
Max. Num. Colonne : 80
Max. Num. Righe : 25
Numero Colori : 2
Numero Pagine : 1
Risoluzione : 640 x 200
Memoria Video (K) : 256

Modo 3
Modo non supportato.

Modo 4
Modo non supportato.

Modo 5
Modo non supportato.

Modo 6
Modo non supportato.

Modo 7
Max. Num. Colonne : 40
Max. Num. Righe : 25
Numero Colori : 16
Numero Pagine : 8
Risoluzione : 320 x 200
Memoria Video (K) : 256

Modo 8
Max. Num. Colonne : 80
Max. Num. Righe : 25
Numero Colori : 16
Numero Pagine : 4
Risoluzione : 640 x 200
Memoria Video (K) : 256

Modo 9
Max. Num. Colonne : 80
Max. Num. Righe : 25
Numero Colori : 16

Numero Pagine : 2
Risoluzione : 640 x 350
Memoria Video (K) : 256

Modo 10
Max. Num. Colonne : 40
Max. Num. Righe : 25
Numero Colori : 16
Numero Pagine : 8
Risoluzione : 640 x 350
Memoria Video (K) : 256

Modo 11
Max. Num. Colonne : 80
Max. Num. Righe : 30
Numero Colori : 2
Numero Pagine : 1
Risoluzione : 640 x 480
Memoria Video (K) : 256

Modo 12
Max. Num. Colonne : 80
Max. Num. Righe : 30
Numero Colori : 16
Numero Pagine : 1
Risoluzione : 640 x 480
Memoria Video (K) : 256

Modo 13
Max. Num. Colonne : 40
Max. Num. Righe : 25
Numero Colori : 256
Numero Pagine : 1
Risoluzione : 320 x 200
Memoria Video (K) : 256

Il modo 3 è supportato se si emula la Hercules e si carica il programma TSR QBHERC.COM prima di qualsiasi altro.

*** SEEK**

Questa istruzione, usata talvolta insieme all'omonima funzione, permette di posizionare un puntatore ad un file in maniera da poterne, in seguito, leggere o scrivere un determinato byte. La sintassi dell'istruzione è

SEEK [**#**]**numfile, posizione**

Il primo parametro rappresenta il numero del file a cui l'istruzione si riferisce, ed il secondo, per i file BINARY, INPUT, OUTPUT e APPEND, è il numero del byte a cui si sposta il puntatore; tale

valore è compreso tra 1 ed il numero 2147483647. Per i soli file RANDOM, tale argomento non è il numero del byte come detto ma il numero del **record** all'inizio del quale viene spostato il puntatore.

Le periferiche standard del BASIC non supportano tale istruzione che, in questo caso, viene ignorata. Notare che per aggiungere dati ad un file che non sia RANDOM, è sufficiente la seguente linea

```
SEEK #numfile, LOF(numfile)+1
```

ed in seguito la scrittura dei dati.
Il seguente esempio mostra come è possibile prelevare una parte qualsiasi di un file e scriverla su un altro; tale programma si comporta, per i file, come la funzione MID$ per le stringhe; è per questo che è stato chiamato FMID

```
' (Main Module FMID.BAS)
DEFLNG A-Z
CLS
INPUT "Nome File in lettura : ", FR$
INPUT "Nome File in scrittura : ", FW$
INPUT "Byte di inizio : ", BY
INPUT "Numero bytes : ", NB
CH$= " "
OPEN "B",1,FR$
OPEN "B",2,FW$
SEEK #1, BY
CNT=NB
DO WHILE CNT>0
        GET #1,,CH$
        PUT #2,,CH$
        CNT=CNT-1
LOOP
CLOSE
END
```

*** SEG**

Vedere CALL, CALLS, DECLARE

*** SELECT CASE**

Questa istruzione, insieme a CASE, CASE ELSE, END SELECT, costituisce una struttura decisionale multipla, molto efficiente e largamente usata in Quick Basic. La sua sintassi completa è

```
SELECT CASE esprcontr
    CASE vallist1
        istruz1
    CASE vallist2
        Istruz2
    …
```

```
        CASE ELSE
            istruz3
    END SELECT
```

in cui esprcontr è un'espressione, numerica o alfanumerica, dal cui valore dipende l'esecuzione o meno di un gruppo di istruzioni che seguono. Infatti, dopo ogni clausola CASE, viene fatta seguire una serie di valori che esprcontr può assumere ed ad ogni CASE corrisponde un blocco di istruzioni che saranno eseguite. Ad esempio,

```
SELECT CASE Iva%
    CASE 4
        PRINT "Iva al 4% = "
    CASE 9
        PRINT "Iva al 9% = "
    CASE 19
        PRINT "Iva al 19% = "
    CASE 38
        PRINT "Iva al 38% = "
    CASE ELSE
        PRINT "Codice Iva sconosciuto."
END SELECT
```

in questo caso, se il contenuto della variabile Iva% è uguale a 19, viene eseguita l'istruzione corrispondente che fa apparire a video la frase "Iva al 19% = "; se il valore di Iva% non dovesse essere compreso tra quelli previsti, vengono eseguite le istruzioni seguenti il CASE ELSE. Il CASE ELSE è comunque opzionale e, quando non viene usato ma dovrebbe esserlo, viene visualizzato da QB 4.0. l'errore CASE ELSE EXPECTED; tuttavia la versione 4.5. di QB, si comporta in maniera differente, non emettendo alcun errore e facendo continuare l'esecuzione del programma **dopo** la clausola END SELECT.
Nell'esempio seguente, quindi

```
K=1024
SELECT CASE K
    CASE 256
        PRINT "Memoria insufficiente."
    CASE 640
        PRINT "Memoria Ok."
END SELECT
```

mentre il QB 4.0. emette il messaggio di errore CASE ELSE EXPECTED, il QB 4.5. non lo fa e continua l'esecuzione dopo l'istruzione END SELECT.

Dopo la clausola CASE è possibile specificare il valore da testare in diverse maniere

- se si vuole usare un operatore relazionale è possibile con la clausola IS scrivere, ad esempio

```
CASE IS < 500
```

oppure

CASE IS >= 23

- se si intende specificare un intervallo di valori piuttosto che un solo valore, si può usare la clausola TO, come

  CASE 34 TO 90

  in cui i valori validi sono rappresentati dall'intervallo 34..90

- è infine possibile specificare in più modi i valori da controllare in un'unica CASE, separando i vari casi con una virgola

  CASE 12, 15 TO 20, IS > 35

  in questo esempio i valori validi saranno 12, quelli nell'intervallo 15..20 e quelli maggiori di 35 (dal 36 in poi)

L'espressione di controllo e i relativi valori in CASE, possono essere delle stringhe facendo attenzione ai seguenti fatti

- il controllo di un intervallo di valori, come ad esempio

  CASE "armadio" TO "valigia"

  viene fatto usando l'ordine alfabetico e considerando buono il valore che segue nell'ordine alfabetico il primo valore specificato e precede l'ultimo;

- le lettere minuscole e le maiuscole, avendo codici ASCII differenti, non sono considerate nello stesso ordine ma, per esempio, si ha che

  armadio > Armadio > ARMADIO

  in quanto le minuscole seguono le maiuscole; è buona norma, nei controlli, usare la funzione **UCASE$** e testare con valori sempre maiuscoli; ad esempio

```
INPUT Parola$
SELECT CASE UCASE$(Parola$)
        CASE "MARITO", "MOGLIE"
                PRINT "Caso 1"
        CASE "FIGLIO", "FIGLIA"
                PRINT "Caso 2"
        CASE ELSE
                PRINT "Caso 3"
END SELECT
```

prestare attenzione al fatto che il contenuto della variabile Parola$, non viene alterato in quanto la conversione in lettere maiuscole viene effettuata in un'area di memoria temporanea solo per l'esecuzione della SELECT CASE.

*** SHARED**

Questa istruzione, che può essere usata solo a livello di Sub o di Function, serve a rendere delle variabili dichiarate a livello di modulo, accessibili anche a una determinata Sub o Function. In questa maniera si evita di dover passare come parametro tutte le variabili che servono alla Sub o alla Function; mentre con l'istruzione COMMON SHARED o DIM SHARED le variabili e gli arrays diventano comuni a **tutte** le Subs e Functions, con la semplice istruzione SHARED all'interno di una Sub o di una Function rende comuni le variabili dichiarate al modulo e **solamente alla Sub o Function** in cui compare l'istruzione. La sua sintassi è la seguente

SHARED variabile[**( )**] [**AS tipo**] [**,…**]

in cui **variabile** può essere un array e in questo caso si fanno seguire le doppie parentesi e **tipo** è uno dei tipi di dati validi per QB (integer, long, single, double, string, string *n ,user).

Ad esempio, mentre in questo caso

```
' (Main Module)
DECLARE SUB PRO( )
CLS
A&=56876
PRINT A&
CALL PRO
END
' (Sub PRO)
SUB PRO
        PRINT A&
END SUB
```

vengono visualizzati i valori 56876 e 0 dato che la variabile all'interno della Sub è considerata locale, basta aggiungere

```
' (Sub PRO)
SUB PRO
        SHARED A&
        PRINT A&
END SUB
```

che le due variabili non sono più considerate come disgiunte e diventa accessibile alla Sub PRO la variabile A& dichiarata nel modulo.

Usando l'istruzione SHARED è possibile dichiarare le variabili comuni con la clausola AS **solo** se così erano state dichiarate nel modulo principale; ad esempio,

```
' (Main Module)
DECLARE SUB PRO( )
DIM VV AS INTEGER, PQ AS DOUBLE
CLS
VV%=3943
PQ#=23.2322#
PRINT VV%, PQ#
CALL PRO
END
' (Sub PRO)
SUB PRO
  SHARED VV AS INTEGER, PQ AS DOUBLE
        PRINT VV%, PQ#
END SUB
```

in caso contrario viene evidenziato il messaggio di errore che avverte che le dichiarazioni nel modulo e nella Sub del tipo delle variabili comuni, devono essere uguali.

Attenzione infine al fatto che, con l'istruzione SHARED, non è possibile far condividere alle Subs o Functions delle variabili dichiarate nelle librerie o in moduli compilati e linkati separatamente da quello principale.

*** SHELL**

Come tramite il menu File è possibile passare in DOS temporaneamente (opzione Shell) richiamando l'interprete dei comandi (COMMAND.COM /C), così è pure possibile durante l'esecuzione di un programma con l'istruzione SHELL, la cui sintassi è la seguente

SHELL [**stringacom**]

Se l'istruzione SHELL viene usata senza argomenti, viene richiamato un nuovo interprete e si passa in DOS; il QB con il programma in esecuzione rimane in memoria, viene visualizzato un messaggio che ricorda di usar il comando EXIT del DOS per abbandonare il nuvo interprete e tornare all'esecuzione del programma Basic all'istruzione seguente la SHELL. L'interprete dei comandi viene ricercato all'interno della directory corrente e, se non viene trovato, tramite la variabile d'ambiente COMSPEC; se il COMMAND.COM non è rintracciabile su disco, viene emesso un segnale di errore dal QB (Illegal function call).

Usando un argomento alfanumerico dopo l'istruzione SHELL, questo viene passato all'interprete dei comandi che lo esegue; viene cercato prima un file con estensione .COM, poi uno .EXE ed infine, se nessuno dei precedenti esiste, un file con estensione .BAT; se neanche quest'ultimo viene trovato (viene anche sfruttata l'indicazione di ricerca della variabile PATH), il DOS emette un errore mentre il QB ignora tale condizione.

Durante l'esecuzione del **processo figlio** (lanciato tramite processore dei comandi secondario), è possibile la redirezione dell'input e dell'output, come nel seguente esempio

```
CLS
```

```
SHELL “DIR | FIND “ + CHR$(34) + “free” + CHR$(34) + “ >TEMP”
OPEN “I”, 1, “TEMP”
INPUT #1, R$
PRINT R$
CLOSE
KILL “TEMP”
END
```

in cui tramite l’istruzione SHELL viene eseguito, dal processore di comandi secondario, il comando DOS

DIR | FIND “free” >TEMP

(notare che la parola **free** deve essere scritta in minuscolo e che le virgolette in cui è racchiusa sono ottenute tramite le due funzioni CHR$(34) del QB); tale comando lista il contenuto di una directory del disco attivo, il cui risultato, con la tecnica del piping (|), viene passato al comando esterno FIND del DOS che ricerca la linea in cui compare la parola free (ultima riga della directory) e la scrive in un file sequenziale temporaneo (TEMP) che, dopo la lettura, viene eliminato. Tramite questo esempio è possibile quindi conoscere il numero dei file presenti nella directory corrente del drive attivo e lo spazio ancora libero su tale disco. Notare come, anche per il processore secondario dei comandi, sia stata sfruttata la possibilità di redirezione e piping di MS-DOS. È importante notare che per potere eseguire il comando FIND che è esterno, questo deve trovarsi all’interno della directory corrente del disco attivo o deve essere raggiungibile tramite la variabile PATH del DOS.

*** SIGNAL**

Questa istruzione, pur essendo presente nell’insieme delle parole chiave di QB 4.5. e di QB 4.0., non viene eseguita ed è riservata a sviluppi futuri. La sua sintassi è

SIGNAL(espr) {**ON** | **OFF** | **STOP**}

In QB 4.0. il suo uso provoca l’emissione di un errore (alquanto strano) che riguarda l’istruzione SELECT CASE.

*** SINGLE**

Vedere AS, COMMON, DECLARE, DEF FN, DIM, FUNCTION, SHARED, STATIC, SUB, TYPE

*** SLEEP**

Questa istruzione, riservata ad usi futuri in QB 4.0., è presente in QB 4.5. e serve a generare una pausa, durante l’elaborazione, di durata definibile. La sua sintassi, molto semplice, è la seguente

SLEEP [**secondi**]

in cui l’unico parametro possibile rappresenta il numero di secondi che la pausa deve durare. Tale pausa termina però anche in conseguenza di altri eventi come

- si preme un tasto sulla tastiera

- si verifica un evento del tipo ON…

Nel programma di esempio, si nota come l'istruzione SLEEP può essere usata per temporizzare con molta precisione dei contatori

```
CLS
TE=3
LOCATE 10,10
PRINT TE
DO WHILE TE>0
      SLEEP 1
      TE=TE-1
      LOCATE 10,10
      PRINT TE
LOOP
```

*** SOUND**

Con questa istruzione è possibile generare dei suoni tramite l'altoparlante del sistema. Anche se non ha la stessa potenza dell'istruzione PLAY, la SOUND è molto spesso utilizzata per la generazione di effetti sonori speciali. La sua sintassi è la seguente

SOUND frequenza, durata

in cui il valore frequenza è misurato in Hertz ed è compreso tra 37 e 32767, anche se questi limiti sono teorici; infatti, già da 1300 Hertz in poi l'altoparlante interno non riesce a generare la frequenza con potenza tale da renderla udibile perché dotato di caratteristiche che non gli consentono di vibrare a tali frequenze.

La durata è invece calcolata in ticks (1 secondo = 18.2 ticks) misura indipendente dal clock del sistema; se tale valore diventa zero, ogni suono generato da una SOUND precedente, viene annullato. Infatti, le istruzioni SOUND sono eseguite in background fino ad un numero di 2 contemporaneamente.

Per generare quindi un 'la' della durata di 1 secondo circa, è necessaria la seguente istruzione

SOUND 440, 18

*** STATIC**

Questa clausola, la cui sintassi è

STATIC nomevar[**( )**][**, …**]

è utilizzata solo nelle Subs, nelle Functions e nelle DEF FN..END DEF, e serve a dichiarare di tipo statico una o più variabile locali. Se dichiarate statiche, le variabili locali conservano il valore all'uscita della funzione o della sub ed usano questo valore se vengono chiamate nuovamente.

Naturalmente una variabile statica non può essere dichiarata SHARED (condivisa con il modulo) perché già definita locale.

L'opzione STATIC usata con l'istruzione SUB e FUNCTION, serve a definire di tipo statico tutte le variabili locali usate all'interno della Function o della Sub; questo fatto non è tuttavia valido per le variabili dichiarate SHARED che prevale nella dichiarazione di tipo delle variabili stesse. L'uso di STATIC nella dichiarazione delle Subs e delle Functions, migliora un po' la velocità di esecuzione delle stesse, dato che la preparazione delle variabili locali non avviene durante l'esecuzione ma durante la compilazione; si ha tuttavia, un utilizzo maggiore di memoria.
Quindi, se una Sub viene dichiarata nel seguente modo

```
SUB PROVA STATIC
    …
END SUB
```

tutte le variabili all'interno di PROVA saranno statiche, mentre, se si modifica la Sub nel seguente modo

```
SUB PROVA STATIC
SHARED X!
    …
END SUB
```

la sola variabile X! sarà definita comune al modulo principale e non di tipo statico. Si possono definire statici anche gli arrays, a condizione che il loro nome venga seguito da una coppia di parentesi tonde. Nel seguente esempio, viene definito di tipo statico un array locale di numeri interi

```
SUB PROVA
    STATIC Tmp%( )
    …
END SUB
```

*** STEP**

Vedere FOR

*** STOP**

Questa istruzione, poco usata in Quick Basic, può essere utile per il test ed il debugging dei programmi. Essa interrompe l'esecuzione del programma e si comporta in maniera differente a seconda che si operi nell'ambiente o con un programma compilato sotto MS-DOS. Infatti, nel primo caso, all'arresto del programma viene visualizzata la riga in cui è presente lo STOP, i file eventualmente aperti rimangono tali e l'esecuzione può riprendere in qualsiasi momento utilizzando il tasto **F5**; nel secondo caso, invece, i file aperti sono tutti chiusi e, dato che il controllo del sistema passa al DOS, non è possibile in alcun modo riprendere l'esecuzione del programma. In quest'ultimo caso, viene visualizzato il numero di linea in cui si è bloccato il programma ma solo a condizione che:

1 – il programma sia stato compilato usando uno degli switch /E, /X o /D

2 – l’istruzione STOP sia posta su una riga numerata

se solo la prima condizione è verificata, tuttavia, appare il numero di linea dell’ultima linea numerata prima di quella su cui è posto lo STOP o, se nessuna linea precedente è numerata, appare 0.

Ad esempio, il programma seguente

```
10 CLS
20 FOR T=1 TO 100
30    PRINT T
40    STOP
50 NEXT T
```

se compilato con BC con uno degli switch di cui al punto 1 ed eseguito da DOS, si blocca visualizzando un messaggio che indica la linea 40 come ultima linea eseguita;

in quest’altro programma, invece

```
10 CLS
    FOR T=1 TO 100
       PRINT T
       STOP
    NEXT T
```

nelle stesse condizioni del precedente, viene visualizzato il messaggio che l’ultima linea eseguita era la 10;

infine in quest’ultimo esempio,

```
CLS
FOR T=1 TO 100
       PRINT T
       STOP
NEXT T
```

compilato sempre con uno degli switch (/D, /E o /X) viene visualizzato il messaggio che l’esecuzione si è arrestata alla linea 0.

L’istruzione STOP può essere messa in qualsiasi punto del programma, anche in più parti, e serve per bloccare il programma al fine di potere controllare il valore intermedio delle variabili. Infatti, il programma può essere bloccato, può essere controllato, in maniera Immediate, il contenuto di una o più variabili, e si può far continuare il programma con il tasto F5; tutto ciò solo se si ci trova in ambiente Quick Basic; è comunque consigliabile usare per il debug, l’apposito menu con tutte le funzioni messe a disposizione da QB per tale scopo.

In maniera Immediate, all'interno dell'ambiente QB, non è possibile usare l'istruzione STOP perché tale uso è privo di senso.

*** STRIG(n)**
Vedere ON STRIG(n)

*** STRING**
Vedere AS, COMMON, DECLARE, DEF FN, DIM, FUNCTION, SHARED, STATIC, SUB, TYPE

*** SUB**
È l'istruzione usata per definire un sottoprogramma all'interno di un modulo.
La sua sintassi è

SUB nome sub[**(elencoparam)**] [**STATIC**]

in cui nome sub è il nome dato alla Sub, di lunghezza non superiore ai 40 caratteri (inizianti per lettera) e che non sia stato utilizzato da altra Sub o Function.

Elencoparam è invece, l'elenco dei possibili parametri che possono essere passati dal programma chiamante e deve rispettare la seguente sintassi

variab[**( )**][**AS tipo**][**, …**]

in cui tipo è uno di quelli consentiti da QB. Se si devono passare come argomenti degli arrays, l'uso delle doppie parentesi ( ) è obbligatorio dopo il nome della variabile.

La clausola STATIC, a cui si rimanda, è usato per dichiarare di tipo statico tutte le variabili che, se non diversamente specificato da istruzioni SHARED, sono di tipo locale.

Per definire la fine della Sub, si usa l'istruzione **END SUB**, che riporta il controllo al programma chiamante.

Se, durante l'esecuzione della Sub, si volesse tornare al programma chiamante in maniera forzata **prima** delle END SUB, è sufficiente usare l'istruzione **EXIT SUB** preparata allo scopo.

Una Sub può essere richiamata con l'istruzione **CALL** o con il suo nome seguito dagli argomenti se dichiarata con **DECLARE**; non può essere in alcun modo eseguita invece, tramite GOSUB o GOTO.

Come esempio, nel seguente programma, viene proposto l'utilizzo di una Sub per la soluzione di una equazione di secondo grado

```
' (Main Module)
DECLARE SUB Equa2(a!, b!, c!)
CLS
INPUT COEA, COEB, COEC
CALL Equa2(COEA, COEB, COEC)
```

```
END
' (Sub Equa2)
SUB Equa2(A, B, C) STATIC
    IF A=0 THEN
        IF B<>0 THEN
            PRINT "Equazione di primo grado"
            X=-(C/B)
            PRINT "Soluzione : "; X
        ELSE
            PRINT "Equazione senza senso"
        END IF
        EXIT SUB
    END IF
    D=(B*B)-(4*A*C)
    IF D<0 THEN
        PRINT "Soluzioni immaginarie"
        EXIT SUB
    END IF
    X1=(-B-SQR(D))/(2*A)
    X2=(-B+SQR(D))/(2*A)
    PRINT "Soluzioni : ", X1; " e "; X2
END SUB
```

*** SWAP**

Questa istruzione serve a scambiare il contenuto di due variabili dello stesso tipo, senza doverne creare una terza di comodo. Infatti, normalmente, per scambiare il contenuto di due variabili, ad esempio, di ripo numerico intero, era necessario crearne una terza

```
A% = 5
B% = 20

K% = A%
A% = B%
B% = K%
```

per non perdere il contenuto di una delle due durante lo scambio. Ma l'istruzione SWAP provvede automaticamente allo scambio, secondo la seguente sintassi

SWAP var1, var2

Le due variabili devono essere dello stesso tipo altrimenti viene generato un errore del QB. È possibile scambiare anche il valore di due records definiti in memoria con un tipo utente; come esempio, consultare il seguente programma

```
' (Main Module)
DECLARE SUB VISFOR( )
TYPE Fornitore
        RagSoc AS STRING * 30
```

```
		PIva AS STRING * 11
		Indir AS STRING * 30
		Locali AS STRING * 20
		Prov AS STRING * 2
		Saldo AS SINGLE
	END TYPE
	DIM SHARED F1 AS Fornitore
	DIM SHARED F2 AS Fornitore
	F1.RagSoc = “Rossi Paolo”
	F1.PIva = “00082734234”
	F1.Indir = “Via dell’Ermellino, 454”
	F1.Locali = “Palermo”
	F1.Prov = “PA”
	F2.RagSoc = “Verdi Enrico”
	F2.PIva = “00045645644”
	F2.Indir = “Viale Venerina, 112”
	F2.Locali = “Roma”
	F2.Prov = “RM”
	CLS
	PRINT “Prima dello scambio …”
	PRINT
	CALL VISFOR
	A$=INPUT$(1)
	SWAP F1, F2
	CLS
	PRINT “Dopo lo scambio …”
	PRINT
	CALL VISFOR
	END
	‘ (Sub VISFOR)
	SUB VISFOR
		PRINT “ Fornitore 1 :”
		PRINT F1.RagSoc
		PRINT F1.PIva
		PRINT F1.Indir
		PRINT F1.Locali
		PRINT F1.Prov
		PRINT
		PRINT “ Fornitore 2 :”
		PRINT F2.RagSoc
		PRINT F2.PIva
		PRINT F2.Indir
		PRINT F2Locali
		PRINT F2.Prov
	END SUB
```

*** SYSTEM**

Questa istruzione, molto usata nel BASICA e nel GWBASIC, è poco utile in QB dato che le sue funzioni sono assolte dall'istruzione END. È comunque riconosciuta e si comporta diversamente a seconda del contesto in cui viene eseguita.

Nell'ambiente QB, in modo Immediate, permette di tornare al sistema operativo chiudendo i file e, se necessario, aggiornando il file del programma in corso di edit; sempre nell'ambiente, ma all'interno di un programma, una volta eseguita si comporta come l'istruzione END, chiudendo i file aperti e facendo tornare il controllo al QB.

Nei programmi compilati ed autonomi, permette la chiusura dei file aperti ed il ritorno al sistema operativo, come l'istruzione END.

Nell'esempio seguente si può vedere come possa essere usata questa istruzione

```
IF COMMAND$="" THEN
    PRINT "Argomenti mancanti. Procedura abortita"
    SYSTEM
END IF
```

*** THEN**

Vedere IF

*** TIME$**

Questa istruzione, talvolta usata con l'omonima funzione, serve a predisporre l'orario attuale. La sintassi è

TIME$ = stringa

in cui 'stringa' contiene l'orario che si vuole impostare in una delle seguenti tre forme

hh
hh:mm
hh:mm:ss

di chiaro significato; i dati mancanti, a seconda della forma prescelta, sono impostati automaticamente a zero (nella versione 4.5. di QB, a differenza della 4.0., stranamente, l'uso della prima forma, senza i minuti e i secondi, provoca un errore; per rimediare, è sufficiente aggiungere i minuti a zero usando la seconda forma). L'orologio è di tipo 24 ore e quindi per le 4 del pomeriggio si deve introdurre il valore 16:00:00.

Attenzione al fatto che l'impostazione dell'orologio con questa istruzione, modifica l'orario conservato dal DOS e modificabile con il comando TIME di MS-DOS.
Il seguente esempio mostra come può essere usata l'istruzione TIME$ in congiunzione con l'omonima funzione per controllare il passare di un determinato tempo

```
CLS
PRINT "Premere il tasto … ";
A$=INPUT$(1)
```

```
PRINT "Ok."
TIME$="00:00:00"
DO WHILE TIME$<>"00:00:02"
LOOP
PRINT "Sono passati 2 secondi dall'uso del tasto."
BEEP
END
```

*** TIMER (controllo gestore eventi)**

Vedere ON TIMER(n)

*** TO**

Vedere FOR

*** TROFF**

Questa istruzione, insieme alla **TRON**, serve ad evidenziare il flusso di esecuzione di un programma per facilitarne il debug. Mentre TRON attiva la visualizzazione passo-passo dell'attività del programm, TROFF la disattiva. In ambiente Quick Basic ha la stessa funzione del comando Trace On del menu Debug; compilato con BC, serve a stampare i numeri di linea che via via vengono eseguite. Per fare ciò però, è necessario compilare il programma con l'opzione **/D**.

Ad esempio, il seguente programma, che stampa i primi 10 numeri interi dal numero 1, visualizza anche i numeri di linea eseguita; è obbligatorio in questo caso numerare le linee da eseguire:

```
10 TRON
20 CLS
30 FOR T=1 TO 10
40     PRINT T
50 NEXT T
60 TROFF
70 END
```

Le caratteristiche di debugging del Quick Basic, rendono superfluo ed obsoleto l'uso di queste due istruzioni che sono state conservate per compatibilità con il BASICA.

*** TYPE**

L'istruzione TYPE, insieme alla END TYPE, definisce un tipo di dato utente derivante dall'unione di più dati primitivi del QB. È simile ai records del PASCAL o alle struct del C e costituisce una delle novità che rende il QB molto più potente di ogni suo predecessore.
La sintassi di questa istruzione è abbastanza articolata

```
TYPE NomeDatoUtente
        Variabile AS TipoDato
        Variabile AS TipoDato
        …
END TYPE
```

Il nome del dato indicato dopo l'istruzione diventa il nome assegnato al nuovo tipo di dato (come è possibile in altri linguaggi) che si va ad aggiungere a quelli primitivi. Il nuovo dato è definito come l'unione di tutti quegli elementi che si trovano tra le due istruzioni TYPE ed END TYPE. Il tipo di dato assegnato ad ogni elemento **può anche essere un tipo utente definito in precedenza** tramite una struttura TYPE..END TYPE, oltre che uno di quelli primitivi (INTEGER, LONG, SINGLE, DOUBLE, STRING * n); solamente le stringhe a lunghezza variabile non possono essere specificate all'interno di tali strutture e sono sostituite da quelle a lunghezza fissa (STRING * n).

La definizione di un tipo di dato utente **non riserva memoria** per i dati da conservare; questa operazione va fatta definendo ogni variabile di tipo utente di cui si ha bisogno, con l'istruzione

DIM Variab **AS** NomeDatoUtente

dopo che la struttura di tale dato è stata precisata. È possibile dimensionare interi arrays di elementi di tipo utente ed indirizzare le singole parti che lo compongono. Per le variabili semplici, gli elementi vanno indirizzati con un punto tra il nome della variabile utente ed il nome dell'elemento preso in considerazione, come nel seguente esempio

```
TYPE Persona
        Nominativo AS STRING * 30
        Eta AS INTEGER
END TYPE
DIM Impiegato AS Persona
Impiegato.Nominativo = "ROSSI PAOLO"
Impiegato.Eta = 45
PRINT Impiegato.Nominativo
PRINT Impiegato.Eta
END
```

Non è possibile indirizzare, in un sol colpo, tutti gli elementi di una variabile utente, sia questa semplice che indicizzata.
Nel caso di array di tipo utente, il precedente programma va modificato nel seguente modo

```
TYPE Persona
        Nominativo AS STRING * 30
        Eta AS INTEGER
END TYPE
DIM Impiegati(1 TO 10) AS Persona
FOR X=1 TO 10
        INPUT Impiegati(X).Nominativo
        INPUT Impiegati(X).Eta
        PRINT
NEXT X
FOR X=1 TO 10
        PRINT Impiegati(X).Nominativo, Impiegati(X).Eta
NEXT X
```

```
END
```

Nel caso in cui fosse necessario, è possibile trattare arrays di variabili di tipo utente i cui elementi sono a loro volta, di tipo utente, come nel seguente esempio

```
TYPE Persona
    Nominativo AS STRING * 30
    Eta AS INTEGER
END TYPE
TYPE Ditta
    RagSoc AS STRING * 30
    NumImp AS INTEGER
    Presidente AS Persona
    Direttore AS Persona
END TYPE
DIM ElencoDitte(1 TO 10) AS Ditta
CLS
FOR X=1 TO 10
    INPUT ElencoDitte(X).Presidente.Nominativo
    INPUT ElencoDitte(X).Presidente.Eta
    PRINT
NEXT X
FOR X=1 TO 10
    PRINT ElencoDitte(X).Presidente.Nominativo;
    PRINT ElencoDitte(X).Presidente.Eta
NEXT X
END
```

È evidente, in questo caso, come sia necessario definire i tipi di dati utente utilizzati nella seconda TYPE..END TYPE **prima** di questa. Tutti i singoli elementi della variabili sono separati dal punto per potere accedere ad ognuno di loro; l'indice, se presente, va comunque posto dopo il nome della variabile definita con l'istruzione DIM.

L'utilizzo principale della struttura TYPE..END TYPE, in Quick Basic, è quello della definizione del record dei file random, metodo che sostituisce l'uso dell'istruzione FIELD, della LSET, RSET e delle funzioni CV.. e MK..$ (vedere quest'ultime per un esempio di utilizzo).

*** UNLOCK**

Vedere LOCK

*** UNTIL**

Vedere DO

*** VIEW**

È una delle istruzioni usate in programmi per la grafica. Essa definisce l'area del video da usare per visualizzare i grafici. La sintassi di VIEW è

VIEW [[**SCREEN**] **(X1, Y1)-(X2, Y2)**[,[**Colore**][,**Bordo**]]]

in cui

- la parola chiave SCREEN è usata per definire le coordinate assolute rispetto allo schermo; se non viene usata, le coordinate sono relative ai bordi della finestra definita;

- X1, Y1, X2 e Y2 sono le coordinate di due angoli opposti dell'area definita dall'istruzione;

- Colore è il colore usato per riempire l'area definita da VIEW; se non è presente, l'area non viene riempita;

- Bordo è il colore del bordo dell'area; se non viene indicato, il bordo non viene evidenziato.

Una volta definita l'area per i grafici con l'istruzione VIEW, ogni elemento grafico viene tracciato solo all'interno della stessa; un eventuale riferimento al di fuori dell'area, non arresta il programma, ma viene ignorato.
Se l'istruzione VIEW viene usata senza argomenti, l'intero schermo viene usato come area di output per la grafica. Anche una istruzione RUN o una SCREEN hanno l'effetto di annullare ogni istruzione VIEW precedente.

Il seguente esempio, mostra come è possibile agire, con le istruzioni LINE e CIRCLE, su un'area definita dallo schermo

```
RANDOMIZE TIMER
CLS
SCREEN 2
LOCATE 5, 12
PRINT “Esempio di uso di VIEW”
VIEW (100, 50)-(250, 120)
COL=1
DO
        LINE (RND*150, RND*70)-(RND*150, RND*70), COL XOR 1
        CIRCLE (RND*150, RND*70), RND*10, COL
        A$=UCASE$(INKEY$)
        IF A$<>”” THEN
                COL=COL XOR 1
        END IF
LOOP WHILE A$<>CHR$(27)
VIEW (100, 50)-(250, 120), 0, 1
LOCATE 11, 21
PRINT “FINE”
END
```

È possibile usare una istruzione WINDOW in congiunzione con una VIEW, in modo da definire delle coordinate logiche per un'area di schermata grafica.

*** VIEW PRINT**

Questa istruzione è usata per definire un intervallo di righe di testo entro cui indirizzare i dati con le istruzioni di visualizzazione. La sua sintassi è

VIEW PRINT [**LineaAlta TO LineaBassa**]

in cui LineaAlta e LineaBassa sono due numeri indicanti le linee dello schermo entro le quali è possibile visualizzare dei dati. L'istruzione CLS 2 può essere usata per cancellare l'area definita con questa istruzione. Se VIEW PRINT viene usata senza argomenti, lo schermo intero viene ridefinito come area di output.

Il seguente esempio mostra come è possibile leggere un file sequenziale da disco e visualizzarne il contenuto all'interno di un'area del video

```
CLS
LOCATE 11, 1
PRINT STRING$(80, 196);
LOCATE 16, 1
PRINT STRING$(80, 196);
OPEN "C:\AUTOEXEC.BAT" FOR INPUT AS #1
VIEW PRINT 12 TO 15
R=1
DO WHILE NOT EOF(1)
        INPUT #1, A$
        PRINT R; ") "; A$
        R=R+1
        K$=INPUT$(1)
LOOP
CLOSE
END
```

*** WAIT**

Questa istruzione, la cui sintassi è

WAIT PortaHw, Andval [**, Xorval**]

è utilizzata per sincronizzare il programma con un segnale hardware presente su una porta di I/O del sistema. Come per la funzione INP e l'istruzione OUT, bisogn conoscere il numero della porta e la funzione che svolge la porta selezionata prima di usare l'istruzione WAIT. Essa infatti, attende finché il risultato dell'input da tale porta, dopo aver fatto una operazione di XOR bit per bit con il terzo argomento (se esiste) ed avere fatto una operazione di AND bit per bit con il secondo argomento, diventa diverso da 0. È **estremamente pericoloso** usare tale istruzione senza conoscere le caratteristiche dell'hardware di sistema dato che essa non può essere interrotta che da un reset del sistema stesso.

Nel seguente esempio, si può vedere come la WAIT può essere usata per controllare se la scheda video genera il segnale di ritraccia verticale, su cui il programma si sincronizza. Questo

segnale è presente sul bit 3 della porta 3DAh (esadecimale) per una scheda a colori o 3Bah (esadecimale) per una scheda monocromatica. L'indirizzo esatto viene ricavato dalla locazione 463h (esadecimale) in cui viene depositato dalle routines del BIOS

```
CLS
DEF SEG=0
VideoReg = PEEK(&H463) + 774
DEF SEG
DO
    LOCATE 10, 10
    PRINT Retrace
    WAIT VideoReg, 8
    LOCATE 10, 10
    PRINT Retrace
    Retrace=(Retrace + 1) MOD 50
    WAIT VideoReg, 8
LOOP
END
```

*** WEND**

Vedere WHILE

*** WHILE**

Insieme all'istruzione WEND costituisce una struttura di controllo iterativa, simile alla struttura DO..LOOP, che permette di ripetere una parte di programma fino a che una determinata condizione logica resta vera. La struttura DO..LOOP è più efficiente e flessibile della WHILE..WEND ma, quest'ultima è stata conservata per compatibilità con le precedenti versioni di Basic. La sintassi della struttura WHILE..WEND è la seguente

WHILE condlog
…
WEND

in cui condlog è rappresentata da una espressione logica il cui risultato viene valutato ogni volta che viene iniziato il ciclo; se questo risultato è vero, vengono eseguite le istruzioni all'interno della struttura ed, alla fine viene eseguita nuovamente la WHILE; altrimenti le istruzioni interne non vengono eseguite ed il controllo passa all'istruzione seguente l'istruzione WEND.

Le strutture WHILE..WEND possono essere nidificate in numero consentito dallo stack ed ogni istruzione WEND corrisponde alla più recente WHILE che è stata aperta. Non è consigliabile saltare con istruzioni del tipo GOTO, GOSUB all'interno delle strutture senza passare dall'istruzione WHILE, altrimenti si può verificare un errore durante l'esecuzione del programma.

Nel seguente esempio, viene mostrato come si può usare la struttura WHILE..WEND per realizzare una routine che attende l'uso di un tasto

```
CLS
```

```
PRINT “Premere un tasto”
WHILE INKEY$ = “”
WEND
PRINT “Tasto premuto”
END
```

*** WIDTH**

L’istruzione WIDTH, usata per dichiarare la larghezza della riga di output su una determinata periferica o file, ha una sintassi diversa a seconda del tipo di funzione effettuata, e cioè

WIDTH [**colonne**][**, linee**]

WIDTH {**# numfile** | **device**}**, larghezza**

WIDTH LPRINT larghezza

La prima sintassi è adottata per cambiare il numero di colonne e di righe dello schermo, anche se i valori adottabili sono limitati dal tipo di scheda video disponibile. In generale, per le colonne è possibile usare il valore 40 o 80, mentre, per le righe, i valori possibili sono 25, 30, 43, 50 o 60. Si possono omettere i valori, ma non contemporaneamente; questo permette di modificare solo il numero delle colonne o solo il numero delle righe visualizzate.
Con la scheda VGA in modalità 640 x 480 (modo 12 con l’istruzione SCREEN), è possibile visualizzare 30 righe per 80 colonne, come mostra il seguente esempio

```
CLS
DO
        SCREEN 12
        WIDTH 80, 30
        COLOR 7
        FOR T=1 TO 30
                LOCATE T, 1
                PRINT “Riga numero “; T;
        NEXT T
        A$=INPUT$(1)
        WIDTH 80, 25
        FOR T=1 TO 25
                LOCATE T, 1
                PRINT “Riga numero “; T;
        NEXT T
        A$=INPUT$(1)
LOOP WHILE UCASE$(A$)<>”F”
SCREEN 0
CLS
END
```

La seconda sintassi è adottata per modificare la larghezza della riga di output di una periferica, sia che questa venga aperta in precedenza (istruzione OPEN), sia che venga specificata

direttamente nella stessa istruzione WIDTH. Ad esempio, per indicare alla stampante che la larghezza della riga di stampa è di 10 caratteri, si può usare il seguente programma

```
CLS
OPEN "LPT1:" FOR OUTPUT AS 1
WIDTH #1, 10
FOR T=1 TO 20
        PRINT #1, STRING$(50, 65);
NEXT T
A$=INPUT$(1)
CLOSE
END
```

oppure, togliendo le istruzioni OPEN e CLOSE, si può specificare il nome della periferica direttamente con la linea

```
WIDTH "LPT1:", 10
```

La terza sintassi è usata per semplificare le operazioni con la stampante; infatti, la parola chiave aggiuntiva LPRINT indica direttamente a quale periferica fare riferimento. L'uso dell'istruzione WIDTH LPRINT … è necessaria quando si deve stampare una riga di dati più lunga di 80 caratteri su una stampante che ha il carrello sufficientemente ampio per 132 caratteri. Il seguente programma di esempio, mostra come è possibile stampare, su una stampante con le caratteristiche suddette, in modo compresso, ben 230 caratteri su una riga

```
WIDTH LPRINT 255
LPRINT CHR$(15);
LPRINT STRING$(230, 45)
LPRINT TAB(110); "Prova di stampa"
LPRINT STRING$(230, 45)
END
```

Notare che il valore massimo che si può impostare con l'istruzione WIDTH LPRINT è 255.

*** WINDOW**

Questa potente istruzione permette di stabilire delle coordinate 'logiche' per la grafica da usare al posto di quelle 'fisiche'. In altre parole, nelle istruzioni grafiche come PSET, PRESET e LINE, è possibile specificare, al posto delle coordinate in pixels, dei valori, presi tra certi limiti, che costituiscono le nuove coordinate. I limiti imposti alle coordinate logiche vanno, appunto, definiti con tale istruzione. La sintassi completa di WINDOW è

WINDOW [[**SCREEN**] **(X1, Y1)-(X2, Y2)**]

in cui X1, Y1, X2 e Y2 sono le coordinate logiche dei due angoli della finestra video definita. Tale finestra, normalmente, coincide con l'intero video, ma le sue dimensioni possono essere mutate con l'istruzione VIEW. Usando l'istruzione WINDOW, la coordinata Y aumenta dal basso verso l'alto, come nel piano cartesiano; se si volesse invertire tale condizione, rispettando le

regole della numerazione dei pixels sullo schermo, è sufficiente usare la parola chiave SCREEN dopo l'istruzione WINDOW.

Per ritornare ad usare le coordinate fisiche al posto di quelle logiche, è sufficiente eseguire l'istruzione WINDOW senza alcun parametro.

Il seguente programma di esempio serve a visualizzare la funzione radice quadrta, dopo aver scelto un intervallo apposito

```
DO
    CLS
    DO
        LOCATE 10, 10
        INPUT "Intervallo : ", X1, X2
        IF X1>=0 AND X2>X1 THEN
            EXIT DO
        END IF
    LOOP
    SCREEN 2
    WINDOW (X1, SQR(X1))-(X2, SQR(X2))
    LINE (X1, 0)-(X2, 0)
    LINE (0, SQR(X1))-(0, SQR(X2))
    FOR X=X1 TO X2 STEP (X2-X1)/1000
        PSET (X, SQR(X))
    NEXT X
    A$=INPUT$(1)
    SCREEN 0
LOOP WHILE A$<>CHR$(27)
END
```

*** WRITE**

L'istruzione WRITE è usata, come la PRINT, per visualizzare dati sullo schermo. La sua sintassi è la seguente

WRITE [**#nf,**][**espr**[**,…**]]

Il primo parametro, se presente, fa in modo che i dati vengano mandati verso il file o la periferica aperti con una precedente istruzione OPEN il cui numero di riferimento è 'nf', invece che al video. I risultati delle espressioni indicate, verranno visualizzati, separati da virgole, senza spazi iniziali o finali ed inclusi tra virgolette, se stringhe. Quindi, le seguenti righe di programma

```
CLS
X=123
Y=3.14
F$="Dati : "
WRITE F$,X,Y
END
```

produrranno il seguente output a video

"Dati : ",123,3.14

Notare che vengono visualizzate le virgolette per le stringhe e le virgole di separazione tra i dati; inoltre i dati numerici **non** sono preceduti da uno spazio.

Questa istruzione viene preferita alla PRINT # quando si devono scrivere dei dati su file sequenziale; infatti, l'istruzione PRINT # non scrive automaticamente separatori tra i dati, a meno di usare una istruzione PRINT # per ogni dato; così, se in seguito si legge il file generato, l'istruzione INPUT # non può lavorare correttamente. La WRITE # invece, scrive automaticamente una virgola tra i dati, come già visto, e questo permette ad una istruzione INPUT # seguente, di riconoscere tutti i dati separatamente.

Nel seguente esempio di scrittura su file, di cui viene riportato il contenuto finale, si può notare come agisce l'istruzione WRITE #

```
CLS
OPEN "DATI.DAT" FOR OUTPUT AS #1
FOR X=1 TO 5
    WRITE #1, X, SIN(X), COS(X)
NEXT X
CLOSE
END
```

(Contenuto del file DATI.DAT)

```
1,.841471,.5402023
2,.9092974,-.4161468
3,.14112,-.9899925
4,-.7568025,-.6536436
5,-.9589243,.2836622
```

*** XOR**

L'operatore logico XOR esegue una operazione tra due valori interi o interi lunghi, bit per bit, rispettando la tavola della verità illustrata di seguito:

1^ Operando	2^ Operando	Risultato
0 (FALSE)	0 (FALSE)	0 (FALSE)
0 (FALSE)	**1 (TRUE)**	**1 (TRUE)**
1 (TRUE)	0 (FALSE)	**1 (TRUE)**
1 (TRUE)	**1 (TRUE)**	0 (FALSE)

L'operatore XOR possiede, come gli altri operatori logici, il più basso livello di priorità in un'espressione logico-aritmetica.

Ecco un esempio di utilizzo dell'operatore XOR, che non viene sfruttato molto in programmazione; il programma fa uso dell'operatore XOR per crittografare e decrittografare frasi e parole secondo una chiave numerica

```
' (Main Module)
DECLARE FUNCTION XCODIF$ (ST$, KY%)
CLS
INPUT "Stringa "; S$
INPUT "Chiave (0-255) "; K%
SC$=XCODIF$ (S$, K%)
PRINT "Stringa codificata      : "; SC$
PRINT "Stringa decodificata   : "; XCODIF$ (SC$, K%)
END
' (Function XCODIF$)
FUNCTION XCODIF$ (ST$, KY%) STATIC
XC$=""
FOR Y=1 TO LEN(ST$)
        XC$=XC$+CHR$(ASC(MID$(ST$, Y, 1)) XOR KY%)
NEXT Y
XCODIF$=XC$
END FUNCTION
```

5.1.2 Le funzioni di Quick Basic

Di seguito vengono elencate le 85 funzioni standard utilizzabili in Quick Basic, in ordine alfabetico, complete di sintassi ed esempi di utilizzazione.

*** ABS**

È una funzione numerica che restituisce il valore assoluto di una espressione numerica. La sua sintassi è

ABS(esprnum)

Il valore ritornato rispetta la seguente tabella

se esprnum>0	ritorna esprnum
se esprnum=0	ritorna 0
se esprnum<0	ritorna -esprnum

Ad esempio, il seguente programma mostra l'utilizzo della funzione ABS

```
CLS
FOR T=-5 TO 10
        PRINT ABS(T)
NEXT T
END
```

*** ASC**

Complementare alla funzione CHR$, la funzione ASC ritorna il valore numerico corrispondente nella tabella ASCII del primo carattere contenuto nella stringa passata come parametro. La sintassi è quindi la seguente

ASC(stringa)

in cui stringa non può essere vuota.

Nel programma d'esempio successivo, è mostrato il risultato fornito dalla funzione ASC

```
A$="Quick Basic V.4.0."
FOR T=1 TO LEN(A$)
        PRINT ASC(MID$(A$, T, 1));
NEXT T
END
```

*** ATN**

È una funzione numerica che restituisce il valore dell'arcotangente del suo argomento. Il risultato fornito è sempre compreso tra -pigreco/2 e +pigreco/2 radianti.
Se l'argomento è espresso in doppia precisione, il risultato è fornito con tale precisione altrimenti il risultato è in semplice precisione.

Ad esempio, si riporta un programma che stampa su carta, una tabella con dei valori della arcotangente da –pigreco a +pigreco a passi pigreco/8

```
CONST PI# = 3.1415926#
CLS
FOR V#=-PI# TO PI# STEP PI#/8
     LPRINT USING "##.#### → ##.####"; V#, ATN(V#)
NEXT V#
END
```

*** CDBL**

Questa funzione è usata per trasformare in doppia precisione il risultato di una espressione numerica. Questa funzione produce lo stesso effetto della conversione che automaticamente avviene quando il risultato di una espressione numerica viene assegnato ad una variabile in doppia precisione.

Il programma di esempio, infatti, dimostra come la precisione del risultato dipenda dalla precisione degli operandi e non dalla conversione da cui si ottiene sempre un valore poco preciso

```
CLS
A=2/3
B#=2/3
C#=2#/3#
PRINT A
PRINT B#
```

```
PRINT C#
PRINT
PRINT CDBL(A)
PRINT CDBL(B#)
PRINT CDBL(C#)
```

Si può notare infatti che il risultato di CDBL(A) è lo stesso ottenuto assegnando a B# il valore dell'espressione calcolata in singola precisione e che il risultato più preciso si ha facendo calcolare l'espressione con operandi in doppia precisione.

*** CHR$**

Questa funzione è usata per ricavare da un valore numerico passato come parametro, il corrispondente carattere secondo la tabella ASCII. La funzione è quindi di tipo alfanumerico ed ha la seguente sintassi

CHR$(esprnum)

in cui esprnum può essere compreso tra 0 e 255.

Attenzione al fatto che molti caratteri, specialmente se trasmessi ad un dispositivo periferico (come una stampante), possono essere interpretati come comandi piuttosto che come caratteri da stampare. È il caso del carattere con codice 15 che viene usato nelle stampanti per selezionare il metodo di stampa compressa, come nell'esempio seguente

```
LPRINT CHR$(15);"STAMPA COMPRESSA";CHR$(18);"STAMPA NORMALE"
```

*** CINT**

Questa funzione è utile quando serve che un valore, in singola o doppia precisione, debba essere convertito in intero, considerando la parte decimale ed arrotondando il risultato in base a quest'ultima.
Infatti, a differenza delle funzione INT e FIX che troncano il risultato non considerando i decimali, CINT arrotonda il valore che si ottiene dalla conversione all'intero più prossimo.

Il range dei valori che possono essere convertiti senza che QB emetta un errore di Overflow, è compreso tra -32768 e 32767.
Il seguente esempio mostra chiaramente le differenze tra le tre funzioni

```
A=1.6
B=-1.6
CLS
PRINT A, INT(A), FIX(A), CINT(A)
PRINT B, INT(B), FIX(B), CINT(B)
```

i risultati saranno

1.6	1	1	**2**
-1.6	-1	-2	**-2**

*** CLNG**

Come per la CINT, questa funzione provvede ad eseguire la conversione del suo argomento arrotondando i decimali e produce in risultato intero lungo. L'uso è simile a quello della CINT, ma il range dell'argomento può andare da -2147483648 a 214783647, altrimenti si incorre in un errore di Overflow.

Nell'esempio seguente, si nota appunto, la più ampia capacità di conversione dovuta al fatto che questo tipo di dato occupa più memoria.

```
CLS
A=1500000.6
PRINT A, CLNG(A)
```

il risultato sarà dunque

1500000.6 **1500001**

*** COMMAND$**

Questa funzione ritorna la linea di comando immessa quando, un programma compilato viene eseguito sotto DOS. Dalla stringa ritornata dalla funzione vengono eliminati gli spazi alla sinistra e alla destra, e i caratteri alfabetici sono convertiti automaticamente in maiuscolo. Quindi, ad esempio, se si eseguisse da DOS un programma compilato in QB, di nome PROVA.EXE, e si specificasse la seguente lina di comando

prova file.dat /x

la funzione COMMAND$ ritornerebbe, all'interno del programma, la stringa

FILE.DAT /X

senza spazi iniziali o finali.

L'utilità operativa di questa funzione all'interno dell'ambiente di Quick Basic è garantita dall'opzione **Modify COMMAND$...** del menu **Run**, tramite la quale si può simulare l'input di una riga di argomenti sulla linea di comando.

Per constatare l'utilità della funzione COMMAND$, è bene esaminare in dettaglio il seguente programma di esempio (FDUMP.BAS) che può essere reso eseguibile sotto DOS, con i comandi

BC FDUMP;
LINK FDUMP;

Se si dovesse eseguire FDUMP all'interno dell'ambiente di QB, si dovrebbe utilizzare l'opzione Modify COMMAND$... per immettere l'argomento di prova prima di eseguire il programma con i tasti Shift F5.

```
CONST FALSE=0, TRUE=NOT FALSE
ON ERROR GOTO ERTRAP
```

```
PRINT
PRINT “File Dump – Vrs. 1.0.”
FILE$=COMMAND$
IF FILE$=”” THEN
	PRINT “Uso : FDUMP nomefile.est …”
	END
END IF
PRINT
EX=FALSE
DO WHILE NOT EX
	PF=INSTR(FILE$, “ “)
	IF PF=0 THEN
		ACTFILE$=FILE$
		EX=TRUE
	ELSE
		ACTFILE$=LEFT$(FILE$, PF-1)
		FILE$=LTRIM$(MID$(FILE$, PF))
	END IF
	PRINT “In visualizzazione … “; ACTFILE$
	CH$=” “
	CH=1
	SKIP=FALSE
	OPEN “B”, 1, ACTFILE$
	IF NOT SKIP THEN
		DO WHILE NOT EOF(1)
			GET #1, ,CH$
			CH$=HEX$(ASC(CH$))
			IF LEN(CH$)=1 THEN
				CH$=”0”+CH$
			END IF
			PRINT USING “\\ “; CH$;
			IF CH=16 THEN
				PRINT
				CH=0
			END IF
			CH=CH+1
		LOOP
	END IF
	CLOSE #1
	PRINT
	PRINT
LOOP
END
ERTRAP:
	IF ERR=64 THEN
		SKIP=TRUE
		PRINT “Errore in apertura di “; ACTFILE$
		RESUME NEXT
```

```
END IF
ON ERROR GOTO 0
```

*** COS**

È una delle funzioni trigonometriche che il QB mette a disposizione. Le altre sono SIN, TAN e ATN (unica funzione trigonometrica inversa). L'argomento è naturalmente numerico ed è espresso in radianti. La precisione del valore ritornato dipende dalla precisione dell'argomento. Nell'esempio di uso della funzione coseno è mostrato come operare la conversione tra gradi e radianti

```
CONST PI=3.141592565#
CLS
PRINT "Gradi   Radianti   Coseno"
PRINT
FOR G=0 TO 90
    RAD=G*PI/180
    PRINT USING " ##   ##.######   ##.######"; G, RAD, COS(RAD)
NEXT G
END
```

*** CSNG**

È una delle funzioni di conversione tra tipi di dati numerici (le altre sono CINT, CLNG, CDBL), che arrotonda il risultato in maniera matematica. È utile quando si debba utilizzare un valore in singola precisione partendo da un risultato in precisione maggiore. Ad esempio, le seguenti linee di programma

```
CLS
A#=3.1415925656
PRINT CSNG(A#)
```

servono a chiarire il funzionamento di CSNG.

*** CSRLIN**

Questa funzione, senza alcun argomento, restituisce il numero della linea su cui è posizionato il cursore. In molti casi è necessario conoscere tale valore, ed il prossimo programma costituisce un esempio dell'uso di tale funzione e della funzione POS(0)

```
DECLARE SUB NUMTOLET(NUM!)
DIM SHARED NTL$(1 TO 10)
FOR N=1 TO 10
    READ NTL$(N)
NEXT N
CLS
FOR N=1 TO 10
    LOCATE ,20
    PRINT N
    NUMTOLET N
    SLEEP 1
```

```
NEXT N
END
DATA UNO,DUE,TRE,QUATTRO,CINQUE,SEI,SETTE,OTTO,NOVE,DIECI
' (Sub NUMTOLET)
SUB NUMTOLET(NUM) STATIC
    R=CSRLIN
    C=POS(0)
    LOCATE 25, 30
    PRINT SPACE$(10);
    LOCATE 25, 30
    PRINT NTL$(NUM);
    LOCATE R,C
END SUB
```

*** CVD**

Vedere CVI

*** CVDMBF**

Vedere CVSMBF

*** CVI**

Questa funzione, insieme alla CVS, CVL e CVD servono a decodificare i valori numerici scritti sui file random quando questi sono stati codificati in precedenza con le funzioni MKI$, MKL$, MKS$ e MKD$. I suffissi delle funzioni indicano la precisione del valore da convertire, secondo la seguente tabella

I	Valori interi	(2 bytes)
L	Valori interi lunghi	(4 bytes)
S	Valori in singola precisione	(4 bytes)
D	Valori in doppia precisione	(8 bytes)

Il numero di bytes indicato è equivalente alla lunghezza della stringa risultante dall'utilizzo delle funzioni MK..$.
Lo scopo, evidente, dell'utilizzo di tali funzioni, è duplice; si utilizza meno spazio per conservare i numeri rispetto al formato ASCII e si standardizza l'occupazione degli stessi indipendentemente dal loro valore.

Le sintassi delle funzioni MK..$ sono le seguenti

MKI$ (vint)
MKL$ (vintl)
MKS$ (vsp)
MKD$ (vdp)

in cui i valori degli argomenti sono della stessa precisione indicata dal suffisso della funzione. Le stringhe ottenute possono essere così scritte all'interno del file random.
Le sintassi delle funzioni CV.. sono invece

CVI (str2)
CVL (str4)
CVS (str4)
CVD (str8)

in cui gli argomenti sono delle stringhe di lunghezza adeguata alla precisione del valore da convertire, preparate con le funzioni MK..$.

Nell'uso di tali funzioni porre attenzione al fatto che la codifica viene fatta con il formato IEEE, differente dal formato Microsoft Binary usato dalle precedenti versioni di Basic (vedere le funzioni CVSMBF, CVDMBF, MKSMBF$ e MKDMBF$).

Un metodo più razionale ed efficiente per memorizzare dati numerici su file random, viene offerto dal QB con le variabili utente (istruzioni TYPE..END TYPE); usare, quando possibile, questo metodo.
Nei prossimi esempi, viene mostrato l'uso delle funzioni e delle variabili utente per trattare dei dati su file random

```
' (Esempio con le funzioni MK..$ e CV..)
CLS
OPEN "DATI" FOR RANDOM AS 1 LEN=48
FIELD #1, 30 AS NOMIN$, 2 AS ETA$, 4 AS STIP$, 4 AS DIST$, 8 AS REF$
PRINT "Scrittura dati …"
FOR X=1 TO 5
        READ N$, E%, S&, D!, V#
        LSET NOMIN$=N$
        LSET ETA=MKI$(E%)
        LSET STIP$=MKL$(S&)
        LSET DIST$=MKS$(D!)
        LSET REF$=MKD$(V#)
        PUT #1, X
NEXT X
PRINT
PRINT "Lettura dati …"
PRINT
FOR X=1 TO 5
        GET #1, X
        PRINT NOMIN$; CVI(ETA$); CVL(STIP$); CVS(DIST$); CVD(REF$)
NEXT X
CLOSE
END
DATA Rossi Paolo,34,1900000,100.2,1.90991222
DATA Verdi Enrico,22,1750000,88.23,2.88328283
DATA Gialli Piero,25,1770000,23.33,2.23982322
DATA Neri Luigi,32,1880000,5.444,1.22828888
DATA Bianchi Giuseppe,22,1500000,33.32,2.99239994

' (Esempio con TYPE..END TYPE, senza funzioni MK..$ e CV..)
```

```
TYPE REC
    NOMIN AS STRING * 30
    ETA AS INTEGER
    STIP AS LONG
    DIST AS SINGLE
    REF AS DOUBLE
END TYPE
DIM Impiegato AS REC
CLS
OPEN “DATI” FOR RANDOM AS 1 LEN=LEN(Impiegato)
PRINT “Scrittura dati …”
FOR X=1 TO 5
    READ Impiegato.Nomin
    READ Impiegato.Eta
    READ Impiegato.Stip
    READ Impiegato.Dist
    READ Impiegato.Ref
    PUT #1, X, Impiegato
NEXT X
PRINT
PRINT “Lettura dati …”
PRINT
FOR X=1 TO 5
    GET #1, X
    PRINT Impiegato.Nomin
    PRINT Impiegato.Eta
    PRINT Impiegato.Stip
    PRINT Impiegato.Dist
    PRINT Impiegato.Ref
NEXT X
CLOSE
END
DATA Rossi Paolo,34,1900000,100.2,1.90991222
DATA Verdi Enrico,22,1750000,88.23,2.88328283
DATA Gialli Piero,25,1770000,23.33,2.23982322
DATA Neri Luigi,32,1880000,5.444,1.22828888
DATA Bianchi Giuseppe,22,1500000,33.32,2.99239994
```

*** CVL**

Vedere CVI

*** CVS**

Vedere CVI

*** CVSMBF**

Questa speciale funzione di conversione è stata aggiunta al Quick Basic per renderlo compatibile con i file di dati creati con precedenti versioni del linguaggio. Infatti, nella codifica dei valori in singola e doppia precisione (con le funzioni MKS$ e MKD$), le vecchie versioni di

Basic adottavano il formato Microsoft Binary e non quello IEEE usato da QB. Per leggere tali file, quindi, questa funzione, insieme alla **CVDMBF**, sono necessarie per la conversione corretta dei dati numerici. Per potere garantire la possibilità di scrivere nel vecchio formato, sono state introdotte anche le funzioni **MKSMBF$** e **MKDMBF$** che codificano i valori in singola e in doppia precisione, rispettivamente in stringhe di 4 o 8 caratteri di lunghezza, secondo il formato Microsoft Binary.
Le funzioni standard (MKS$, MKD$, CVS, CVD) del Quick Basic possono quindi lavorare solo su file creati con tale linguaggio.

Il seguente programma dimostra come è possibile leggere un file di dati scritto da un programma nel formato Microsoft Binary

```
10 REM Questo programma e’ scritto
20 REM    ed eseguito in GW-Basic
30 CLS
40 OPEN “R”, 1, “DATI.DAT”,100
50 FIELD #1,4 AS V1$,8 AS V2$,88 AS C$
60 X=1234.123
70 X#=5678.567#
80 LSET V1$=MKS$(X)
90 LSET V2$=MKD$(X#)
100 PUT #1,1
110 CLOSE
120 END

‘ (Questo è il programma che legge i dati
‘ scritti dal precedente e li converte)
CLS
OPEN “DATI.DAT” FOR RANDOM AS 1 LEN=100
FIELD #1, 4 AS V1$, 8 AS V2$
GET #1, 1
PRINT “Conversione senza correzione”
PRINT CVS(V1$)
PRINT CVD(V1$)
PRINT
PRINT “Conversione con correzione”
PRINT CVSMBF(V1$)
PRINT CVDMBF(V1$)
CLOSE
END
```

Notare l’errore compiuto convertendo le stringhe in valori numerici con le funzioni standard (CVS e CVD).

Per usare il Quick Basic senza problemi, è consigliabile scrivere delle procedure che riscrivano i file dati convertendo i dati tra i due formati; per fare ciò basta leggere tutti i dati usando le funzioni CVSMBF e CVDMBF e riscriverli usando le funzioni MKS$ e MKD$.

*** DATE$**

Questa funzione, insieme all'omonima istruzione, serve a trattare la data di sistema. Essa ritorna tale data come stringa nel formato **mm-gg-aaaa**, ad esempio con la seguente istruzione

```
PRINT DATE$
```

Il risultato di questa funzione può essere trattato per convertire il formato in **gg-mm-aaaa** più vicino allo standard europeo, con le seguenti linee di programma

```
D$=MID$(DATE$,4,3)+LEFT$(DATE$,3)+RIGHT$(DATE$,4)
PRINT D$
```

*** ENVIRON$**

Questa funzione ritorna il contenuto dell'ambiente del DOS. Nella tabella delle stringhe di ambiente del DOS, vengono conservate, ad esempio, la stringa PATH, che indica al DOS gli eventuali cammini da esaminare nella ricerca dei file eseguibili; o, anche, la variabile COMSPEC, che specifica il cammino ed il nome dell'interprete dei comandi attuale, che normalmente è il COMMAND.COM.
La funzione ha due sintassi, che sono le seguenti

ENVIRON$ (esprstringa)

ENVIRON$ (n)

La prima riga viene usata quando si vuole ottenere il contenuto di una determinata stringa d'ambiente, il cui nome viene passato come argomento alla funzione stessa. Ad esempio, il seguente programma, dopo aver determinato il contenuto della stringa di ambiente PATH, separa e visualizza tutti i cammini in questa specificati

```
CLS
IN=1
E$=ENVIRON$(“PATH”)
X=1
DO WHILE X>0
    X=INSTR(IN, E$, “;”)
    IF X>0 THEN
            PRINT MID$(E$, IN, X-IN)
            IN=X+1
    ELSE
            IF LEN(E$)>X THEN
                    PRINT MID$(E$, IN)
            END IF
    END IF
LOOP
```

Se la variabile PATH non dovesse esistere nell'ambiente, la funzione ENVIRON$ ritornerebbe una stringa vuota, come se, usando la seconda sintassi, la stringa numero n non dovesse essere presente nella tabella dell'ambiente.

Per mezzo della seconda sintassi, si potrebbe visualizzare, ad esempio, il contenuto di tutta la tabella d'ambiente, senza conoscere il nome delle variabili in essa contenute

```
CLS
E=1
DO WHILE ENVIRON(E$)<>""
    PRINT ENVIRON$(E)
    E=E+1
LOOP
```

*** EOF**

Questa funzione ritorna il valore -1 (vero) se viene riconosciuta la fine di un file in uso, altrimenti ritorna 0 (falso). Nella sua sintassi

EOF (nf)

il parametro nf è il numero associato al file nell'istruzione OPEN che apre lo stesso; è quindi possibile testare la fine di **ogni** file aperto in un determinato istante.

Tenere presenti, comunque, le seguenti regole per effettuare un corretto test della fine di un file

- se il file è **sequenziale** è necessario eseguire la funzione EOF **prima** di usare una istruzione di lettura del file stesso; questo per evitare che l'istruzione di lettura, essengo già finito il file, generi un errore di "Input past end"; di regola, il codice che esegue la lettura di un file sequenziale, dovrebbe essere simile al seguente

```
…
DO WHILE NOT EOF(1)
        INPUT #1, X
        …
LOOP
CLOSE #1
…
```

  così facendo, infatti, il test della fine del file viene fatto all'inizio del ciclo DO WHILE; se la funzione EOF ritorna 0 (falso), l'espressione logica NOT EOF(1) ritorna -1 (vero) ed il ciclo viene eseguito una volta con conseguente lettura di un dato numerico nella variabile X; appena finisce il file ed EOF(1) ritorna -1 (vero), l'espressione NOT EOF(1) do come risultato 0 (falso), il ciclo non viene eseguito e il programma continua dopo l'istruzione LOOP con conseguente chiusura del file.

- se il file è **random** o **binario**, invece, il test va effettuato **dopo** l'istruzione GET usata per leggere dato che, se questa non può leggere dati perché oltre la fine del file, solo a questo punto la funzione EOF

ritorna il valore -1 (vero); il codice usato quindi per leggere questi file, è di questo tipo

```
…
GET #1, NR
IF EOF(1) THEN
        CLOSE #1
        END
END IF
…
```

Per quanto riguarda i file random, peraltro, è meglio usare la funzione LOF per controllare l'accesso ai records, invece che la EOF.

Se si usa la funzione EOF per controllare un canale di comunicazione seriale (OPEN "COM"), la funzione EOF ritorna differenti valori a seconda delle varie condizioni in cui viene usata; in particolare

- se il file viene apero in modo **ASCII**, la funzione EOF ritorna falso finché non viene ricevuto il carattere che ha codice ASCII 26 (1B esadecimale, End-of-File mark); a questo punto, la funzione non cambia il valore ritornato se prima non viene chiuso il file;

- se il file viene aperto in modo **binario**, la funzione EOF ritorna falso se il buffer di ricezione è vuoto (e la funzione LOC ritorna zero), altrimenti ritorna vero; il valore di EOF, in questo caso, è continuamente influenzata dalla presenza di caratteri nel buffer di ricezione; in questo caso, è più conveniente usare la funzione LOC.

*** ERDEV**

Questa funzione, insieme alla **ERDEV$**, provvede a ritornare il codice ed il nome della periferica che ha generato un errore critico (INT 24H del DOS). Anche se poco usate, perché sostituite in maniera adeguata dalla funzione ERR, sono utili in certe situazioni che, essendo molto particolari, possono essere ignorate nella normale programmazione.

Le due principali caratteristiche di queste funzioni sono:

- esse sono aggiornate, al verificarsi di un errore su periferica dalla routine di gestione degli errori critici del DOS (INT 24h) che viene controllata da QB; i codici ritornati da ERDEV, quindi, sono quelli dell'interrupt 24h del DOS, mentre il nome della periferica che genera l'errore, è quello assegnato dal DOS;

- il codice d'errore ed il nome della periferica che lo ha generato non sono annullati dopo che sono stati controllati; le due funzioni infatti, continuano a ritornare gli ultimi codici relativi all'ultimo errore verificato; è buona norma, quindi, usare le due funzioni **solo** all'interno

di una routine di gestione degli errori controllata dall'ON ERROR GOTO…

Attenzione al fatto che il codice d'errore 0 **è valido**; tale valore indica che non si è verificato alcun errore **solo** se il nome della periferica, ottenibile da ERDEV$, non è specificato.
L'elenco dei codici ottenibili, con i relativi errori, è il seguente

0 Violazione della protezione da scrittura
1 Numero di drive non valido
2 Il drive non è pronto
3 Comando non valido al disk-controller
4 Error di CRC
5 Lunghezza della 'request structure' errata
6 Errore di seek
7 Formato di disco non riconosciuto
8 Il settore non è stato trovato
9 Fine della carta nella stampante
10 Errore in scrittura
11 Errore in lettura
12 Errore generale

Al codice di errore ritornato da ERDEV, viene aggiunto il valore 8000 esadecimale (-32768 decimale), se l'errore stesso è stato generato da una periferica di tipo carattere; se la periferica è di tipo a blocchi, non viene sommato altro valore al codice di errore. Le periferiche a blocchi sono tutti i drive per dischi mentre, quelle a carattere, sono la stampante, la tastiera e il monitor; anche la porta seriale (comunicazioni seriali) è una periferica di tipo a caratteri.

Ad esempio, il seguente programma dimostra l'uso delle due funzioni considerate

```
ON ERROR GOTO ERTRAP
CLS
LPRINT
PREND:
END
ERTRAP:
    PRINT HEX$(ERDEV), ERDEV$
    RESUME PREND
```

Se la stampante è spenta, esso visualizza la riga

```
800A    LPT1
```

in cui il codice 800A vuol dire "errore numero 10 (0A esadecimale) su periferica di tipo a carattere (al codice viene sommato il valore 8000 esadecimale) di nome LPT1 (la prima stampante parallela per il DOS).

Con il seguente programma, si può invece notare un'altra caratteristica della funzione ERDEV

```
ON ERROR GOTO ERTRAP
CLS
OPEN “A:PROERD” FOR OUTPUT AS #1
PRINT #1, “Prova”
CLOSE #1
PREND:
END
ERTRAP:
        PRINT HEX$(ERDEV), ERDEV$
        RESUME PREND
```

infatti, se il disco nell'unità A è protetto da scrittura, viene visualizzata la seguente riga

0   A:

che indica errore 0 (violazione della protezione da scrittura) su unità a blocchi di nome A: (nome del drive A per il DOS).
Se, invece, il disco non è inserito nel drive A, viene visualizzato

2   A:

(errore di 'Drive non pronto' su unità a blocchi di nome A:).

*** ERDEV$**

Vedere ERDEV

*** ERL**

Questa funzione, in conseguenza di un errore verificatosi durante l'esecuzione di un programma, riporta il numero della riga dove questo si è verificato; in mancanza di questa, viene riportato il numero dell'ultima riga eseguita che ne aveva uno, altrimenti viene ritornato zero.

Ad esempio, il seguente programma

```
ON ERROR GOTO ERTRAP
CLS
100 LPRINT
EREND:
END
ERTRAP:
        PRINT “Errore”; ERR; “alla linea”; ERL
        RESUME EREND
```

se la stampante è spenta, visualizza la frase

Errore 25 alla linea 100

*** ERR**

È la funzione più usata per il controllo degli errori che si verificano durante l'esecuzione dei programmi. Essa viene usata nelle routines di gestione degli errori controllate dall'istruzione ON ERROR GOTO …

Il codice ritornato dalla funzione ERR, varia da 0 a 255 e, per quanto riguarda l'elenco dei possibili errori, le cause di quest'ultimi ed i rimedi da adottare, si rimanda all'appendice B.

Il recupero dell'errore, in certi casi, può essere impossibile senza la modifica del programma sorgente, come nel seguente

```
ON ERROR GOTO ERTRAP
CLS
FOR T=1 TO 5
        READ X
NEXT T
END
ERTRAP:
        PRINT ERR
        RESUME NEXT
        DATA 1,2,"3",4,5
```

Infatti, alla lettura del terzo dato nella frase DATA, viene emesso un 'errore di sintassi' che non è recuperabile dal RESUME NEXT dato che il dato alfanumerico non può essere, in alcun modo, letto dall'istruzione READ in una variabile numerica. In questo caso il codice sorgente va corretto e l'ultima riga va riscritta

```
DATA 1,2,3,4,5
```

*** EXP**

È al funzione inversa di LOG. Essa calcola il valore di 'e' (numero di Nepero) elevato ad x, in cui x è l'argomento della funzione stessa. La sintassi è la seguente

EXP (x)

La precisione del risultato dipende da quella usata per l'argomento. Il massimo valore che è possibile usare per x è 88.02969 in singola precisione, mentre è 709.78271# in doppia precisione. Valori più alti dei suddetti generano un errore di overflow.

Come esempio, è utile esaminare il seguente programma che mostra la relazione esistente tra le funzioni LOG ed EXP

```
CLS
PRINT "Logaritmo in base 10     Valore"
PRINT
FOR L=1 TO 4
        PRINT TAB(10); L, , EXP(L*LOG(10))
NEXT L
END
```

*** FILEATTR**

Questa funzione ritorna delle informazioni riguardanti un file aperto. La sua sintassi

FILEATTR (nf, v)

prevede due valori come argomenti; il primo è il numero del canale associato all'istruzione OPEN con cui è stato aperto il file di cui si vogliono le informazioni; il secondo valore, è un numero, da 1 a 2, il cui significato è il seguente

1 se viene usato questo valore come secondo argomento, la funzione ritorna un codice che indica il modo di apertura del file secondo la seguente tabella

1	INPUT
2	OUTPUT
4	RANDOM
8	APPEND
32	BINARY

2 se viene usato quest'altro valore, viene ritornato dalla funzione il numero interno del DOS di riferimento per il file aperto (DOS handle); questo numero **non è** quello specificato nell'istruzione OPEN di QB.

Il file handle può essere utilizzato per ottenere, tramite la funzione 4400h dell'interrupt 21h del DOS, l'informazione circa il tipo di periferica su cui è stato aperto il file; più precisamente, si può sapere se la periferica in questione è di tipo a blocchi o a carattere. Il seguente programma di esempio effettua tale controllo

```
' (Main Module FATTR.BAS)
' $INCLUDE: 'QB.INC'
CLS
DIM SHARED REG AS REGTYPE
PF$="LPT1:"
OPEN PF$ FOR OUTPUT AS #1
REG.Ax=&H4400
REG.Bx=FILEATTR(1, 2)
INTERRUPT &H21, REG, REG
PRINT PF$; " = Periferica di tipo ";
IF (REG.Dx AND &H80) / &H80 = 1 THEN
        PRINT "carattere"
ELSE
        PRINT "a blocchi"
END IF
CLOSE
END
```

La variabile PF$ contiene il nome della periferica da controllare e, in questo caso, il tipo è "a carattere" (stampante parallela); se la quinta linea viene cambiata con

PF$="C:\PROVA"

la periferica viene indicata "a blocchi" (disco fisso).

Per eseguire questo esempio nell'ambiente QB, bisogna caricare la quick library QB.QLB con il comando

QB /LQB

oppure compilare lo stesso, sotto MS-DOS, servendosi della libreria QB.LIB, con i comandi

BC FATTR;

LINK FATTR,,NUL,QB;

ed eseguirlo richiamando il file FATTR.EXE così ottenuto da DOS.

*** FIX**

Questa funzione ritorna, come la INT, il valore intero dell'argomento. La sua sintassi è

FIX (n)

La differenza che esiste con la INT è minima e si riferisce al tipo di arrotondamento fatto se n è negativo. Infatti, mentre la funzione INT approssima il valore di n al primo intero più piccolo di n, la funzione FIX lo approssima al primo intero più grande di n. Fare attenzione che, per i numeri negativi, il valore più piccolo tra due è quello che valore assoluto maggiore e che l'arrotondamento è in realtà un troncamento.

Per esempio, ecco alcune linee di programma e i risultati corrispondenti, che illustrano le differenze tra FIX ed INT

```
CLS
PRINT FIX(-9.8), INT(9.8)
```

-9 -10

*** FRE**

Questa funzione ritorna ritorna il numero di bytes di memoria RAM disponibili per la conservazione di dati. Le due sintassi della funzione sono le seguenti

FRE(esprstringa)

FRE(esprnum)

in cui esprstringa può essere una qualsiasi espressione che ritorni una stringa, anche nulla, mentre, esprnum è un'espressione numerica che ritorna uno dei seguenti valori, con le relative conseguenze

-1 il numero di bytes disponibili per un array che non sia di stringhe;

-2 il numero di bytes liberi nello stack.

Un qualsiasi altro valore ha come effetto la restituzione del numero di bytes disponibili per le stringhe; allo stesso modo, nella prima sintassi, l'uso di una stringa come argomento permette di conoscere lo spazio libero per le stringhe ma, in questo caso, prima di calcolare tale spazio, le stringhe vengono compattate per occupare il minore spazio possibile.

È bene ricordare che lo stack è un'area di memoria che viene usata per il passaggio di parametri nella chiamata delle Subs e delle Functions. Il numero di bytes liberi disponibili in tale area può essere aumentato tramite l'istruzione CLEAR.

Ecco un esempio di uso della funzione FRE e dei valori da essa riportati (il programma stesso è stato compilato in DOS con il comando **BC FRE;** e **LINK FRE;** nell'ambiente QB l'output sarebbe stato differente; per il numero e le dimensioni del programma TSR che ognuno potrebbe avere caricato, l'output è indicativo perché molto variabili da caso a caso)

```
' $DYNAMIC
DECLARE SUB PRO(A!, B!, C!)
CLS
PRINT "All'inizio del programma …"
PRINT FRE(0), FRE(-1), FRE(-2)
X$="Questa è una prova."
PRINT "Dopo aver usato una stringa …";
PRINT FRE(0), FRE(-1), FRE(-2)
DIM K%(1 TO 350)
PRINT "Dopo aver usato un array …";
PRINT FRE(0), FRE(-1), FRE(-2)
CALL PRO(A!, B!, C!)
X=SETMEM(-2048)
PRINT "Dopo aver sottratto memoria …"
PRINT FRE(0), FRE(-1), FRE(-2)
Y=SETMEM(2048)
PRINT "Dopo aver riallocato memoria …"
PRINT FRE(0), FRE(-1), FRE(-2)
CLEAR ,, FRE(-2)-1000
PRINT "Dopo aver diminuito lo stack …"
PRINT FRE(0), FRE(-1), FRE(-2)
' (Sub PRO)
SUB PRO(A!, B!, C!)
    PRINT "Dopo aver usato una SUB …";
    PRINT FRE(0), FRE(-1), FRE(-2)
END SUB
```

output del programma

All'inizio del programma	…	59528 468744 1502
Dopo aver usato una stringa	…	59506 468722 1502
Dopo aver usato un array	…	59506 468018 1502
Dopo aver usato una SUB	…	59506 468018 1502
Dopo aver sottratto memoria	…	59506 465970 1502
Dopo aver riallocato memoria	…	59506 468018 1502
Dopo aver diminuito lo stack	…	60690 469906 438

*** FREEFILE**

La sintassi di questa funzione non prevede alcun argomento; essa ritorna il prossimo numero che è possibile assegnare ad un file con l'istruzione OPEN. È molto importante usare questa funzione all'interno di Subs o Functions in modo che il numero dei file eventualmente usati all'interno di esse, non sia in contrasto con quelli usati dai moduli che lo utilizzeranno.

Ecco un esempio di come è possibile utilizzare tale funzione all'interno di una Function; la funzione FILEEXIST, infatti, ritorna il valore -1 (vero) se un file, dato come argomento, esiste nella directory corrente, altrimenti 0 (falso); viene fornito anche un modulo di prova della funzione stessa

```
DECLARE FUNCTION FILEEXIST!(Name$)
CLS
PRINT FILEEXIST("BPLUS.QLB")
END
FUNCTION FILEEXIST(Name$)
        NF=FREEFILE
        OPEN "B", NF, Name$
        LN=LOF(NF)
        CLOSE NF
        IF LN>0 THEN
                FILEEXIST=-1
        ELSE
                KILL Name$
                FILEEXIST=0
        END IF
END FUNCTION
```

*** HEX$**

È la funzione usata, in certi programmi, per la trasformazione dei valori numerici dal sistema decimale a quello esadecimale. Essa ritorna una stringa, di lunghezza variabile, che è la rappresentazione nel sistema esadecimale del valore decimale passato come argomento. La sua sintassi è

HEX$(n)

in cui n è il valore decimale da trasformare in esadecimale.

Il valore n è automaticamente convertito in intero ed arrotondato se è decimale; vengono gestiti gli interi lunghi (interi con segno a 32 bits). Il seguente esempio dimostra come è possibile convertire valori numerici dal sistema decimale all'esadecimale (tramite la funzione suddetta) e al binario (tramite una funzione utente)

```
DECLARE FUNCTION BIN$(V&)
CLS
FOR X&=0 TO 20
        PRINT X&, HEX$(X&), BIN$(X&)
NEXT X&
END
FUNCTION BIN$(V&)
        VC&=V&
        BIN2OUT$=""
        DO
                D&=INT(VC&/2)
                R=VC& MOD 2
                BIN2OUT$=MID$(STR$(R), 2) + BIN2OUT$
                VC&=D&
        LOOP WHILE D&>0
        BIN$=BIN2OUT$
END FUNCTION
```

*** INKEY$**

Questa funzione ritorna una stringa, di lunghezza variabile a 0 a 2 caratteri, che costituisce un codice del tasto premuto al momento della sua esecuzione. Se la lunghezza di tale stringa è 0, nessun tasto è stato premuto; se è 1, questa rappresenta il carattere corrispondente al tasto premuto secondo il codice ASCII; se la stringa ritornata è lunga 2 caratteri, il primo ha sempre codice ASCII uguale a zero, mentre il secondo è il numero di scansione della tastiera del tasto premuto.

La funzione non emette alcuna eco verso il video e, perciò, rappresenta un ottimo strumento per il controllo da tastiera dei programmi. Il seguente esempio è utile perché fornisce il modo per conoscere i codici di scansione dei vari tasti della tastiera posseduta

```
CLS
K=0
DO WHILE K<>27
    K$=INKEY$
    SELECT CASE LEN(K$)
            CASE 0
            CASE 1
                    K=ASC(K$)
                    PRINT "ASCII -> "; K
            CASE 2
                    S=ASC(RIGHT$(K$, 1))
                    PRINT "Scan -> "; S
    END SELECT
```

```
LOOP
END
```

*** INP**

La funzione INP serve a ricavare il valore attuale restituito da una porta hardware di I/O. L'uso di questa funzione, la cui sintassi è

INP(n)

è limitato dal fatto che è molto legata all'hardware del sistema su cui viene eseguita. L'esempio dell'uso di tale funzione fa riferimento alla porta 62h (esadecimale) che ritorna un valore ogni volta che viene premuto o rilasciato un tasto della tastiera. È utile notare come questo programma rilevi l'uso del tasto centrale del tastierino numerico quando non è abilitato il Num Lock, operazione altrimenti non effettuabile; infatti, il programma di esempio riguardante la funzione INKEY$, non permette la rilevazione dell'uso di tale tasto

```
CLS
P=&H62
K=INP(P)
PRINT K
DO
    Z=INP(P)
    IF Z<>K THEN
        PRINT Z
        IF K=76 AND Z=204 THEN
            K$=INKEY$
            IF K$="" THEN
                PRINT "(Premuto tasto centrale)"
            END IF
        END IF
        K=Z
    END IF
LOOP
END
```

*** INPUT$**

La funzione INPUT$ ritorna una stringa di n caratteri letti da un dispositivo di input. La sintassi di questa funzione è

INPUT$(n [**,**[**#**]**fn**]**)**

in cui il primo argomento è il numero di caratteri da leggere dal dispositivo mentre il secondo, opzionale, è il numero del file aperto in precedenza se i caratteri devono essere letti da disco; in mancanza di quest'ultimo argomento, i caratteri vengono letti dal dispositivo di input standard (normalmente la tastiera); questo, comunque, può essere rediretto da DOS usando i simboli < > e | per il piping. La funzione INPUT$ attende l'ingresso dei caratteri, a differenza della INKEY$.

L'esempio mostra come è possibile, tramire la re direzione ed il piping, usare la funzione INPUT$ per realizzare un programma 'filtro' come il MORE di MS-DOS; il prossimo programma infatti, visualizza i caratteri in ingresso dopo averli trasformati in maiuscolo (solo per i file ASCII)

```
' (Main Module di UPPER.BAS)
CONST CTRLZ=26, LINEFEED=&HA, RET=&HD
DO
    CH=ASC(INPUT$(1))
    IF CH=RET THEN
        CH2=ASC(INPUT$(1))
        SELECT CASE CH2
            CASE CTRLZ
                PRINT CHR$(CH);
                CH=CTRLZ
            CASE LINEFEED
                PRINT CHR$(CH2);
            CASE ELSE
                PRINT CHR$(CH); UCASE$(CHR$(CH2));
        END SELECT
    ELSE
        IF CH<>CTRLZ THEN
            PRINT UCASE$(CHR$(CH));
        END IF
    END IF
LOOP WHILE CH<>CTRLZ
END
```

Per l'uso di tale programma, si deve compilare il sorgente, sotto MS-DOS, con le linee

BC /D UPPER;

LINK UPPER;

ed eseguirlo, ad esempio, con la linea

DIR | UPPER

che provvede a visualizzare la directory in maiuscolo.
Lo switch /D nella compilazione di UPPER.BAS è necessario per permettere l'interruzione della visualizzazione tramite Ctrl-Break quando richiesto.

*** INSTR**

È una funzione che restituisce un valore numerico dipendente dalla posizione di una stringa all'interno di un'altra. La sintassi di INSTR è

INSTR([inizio,] esprstr1, esprstr2)

in cui esprstr2 è una espressione stringa che viene ricercata all'interno dell'altra espressione stringa esprstr1. Se non viene specificato il numero del carattere d'inizio (primo argomento opzionale), la stringa viene cercata dal primo carattere di esprstr1.
I valori ritornati dalla funzione, in base ai presupposti ed ai risultati della ricerca, sono i seguenti

1)	**numero del carattere di esprstr1 dal quale inizia esprstr2**	se esprstr2 è trovata all'interno di esprstr1;
2)	**0**	se esprstr2 non è stata trovata all'interno di esprstr1;
3)	**0**	se esprstr1 è una stringa nulla;
4)	**0**	se il parametro 'inizio è stato specificato ed è maggiore della lunghezza di esprstr1;
5)	**1**	se il parametro 'inizio' non è stato specificato ed esprstr2 è nulla;
6)	**il valore di 'inizio'**	se il parametro 'inizio' è stato specificato ed esprstr2 è nulla;

Ecco un esempio di ogni situazione suddetta

1) PRINT INSTR("Nuova Inghilterra", "uova")
(restituisce il valore 2)

2) PRINT INSTR("Essere o non essere", "avere")
(restituisce il valore 0)

3) PRINT INSTR("", "quick basic")
(restituisce il valore 0)

4) PRINT INSTR(10, "Essere", "avere")
(restituisce il valore 0)

5) PRINT INSTR("Quick Basic", "")
(restituisce il valore 1)

6) PRINT INSTR(3, "Via Roma", "")
(restituisce il valore 3)

Porre attenzione al fatto che la funzione INSTR è sensibile al fatto che i caratteri siano maiuscoli o minuscoli; per evitare problemi di questo tipo, usare le funzioni UCASE$ o LCASE$ prima di usare la INSTR.

Il seguente esempio mostra come usare l'argomento 'inizio' per separare diverse stringhe da un'unica stringa madre

```
CLS
REC$="Rossi*Paolo*Via Roma 1244*90100*Palermo*"
Campo=1
Inizio=1
DO
	PosTok=INSTR(Inizio, REC$, "*")
	PRINT USING "#) "; Campo;
	PRINT MID$(REC$, Inizio, PosTok-Inizio)
	Inizio=PosTok+1
	Campo=Campo+1
LOOP WHILE Inizio<LEN(REC$)
```

*** INT**

È la funzione che ritorna il valore intero dell'argomento specificato. La sua sintassi è

INT(n)

in cui n è l'argomento numerico della funzione.
La funzione INT **non arrotonda** la parte decimale dell'argomento, ma la tronca; usare la funzione CINT per avere il valore arrotondato dell'argomento.

Nell'esempio seguente

```
CLS
PRINT "   N", "INT(N)"
PRINT
FOR X=1 TO 6
	READ N
	PRINT N, INT(N)
NEXT X
END
DATA 9.1, 9.5, 9.8, -9.1, -9.5, -9.8
```

il cui risultato sullo schermo è

```
   N    INT(N)
 9.1      9
 9.5      9
 9.8      9
-9.1    -10
-9.5    -10
-9.8    -10
```

dimostra come, per valori positivi, la funzione INT ritorna la parte intera dell'argomento troncando i decimali, mentre, per i negativi, ritorni, sempre troncando i decimali, il numero intero più piccolo seguente.

*** IOCTL$**

Questa funzione, usata raramente, è destinata all'input di dati derivanti da un device driver installato all'atto del boot del DOS. La sintassi è la seguente

IOCTL$(ncan)

Il parametro della funzione è il numero del canale associato, nell'istruzione OPEN, alla periferica in questione. La stringa ritornata dipende, in modo esclusivo, dal tipo di device driver installato.
La funzione IOCTL$ può svolgere la propria attività solo se esistono le seguenti condizioni

1) è stato installato correttamente il device driver a cui si riferisce l'istruzione OPEN;

2) il device driver in questione può usare le stringhe in modo IOCTL; non tutti i drivers infatti, le gestiscono.

Le periferiche standard del DOS ed i drive **non** supportano le stringhe IOCTL; non è quindi usabile tale funzione con questi dispositivi.
Tenere presente che, anche se si aprisse, con l'istruzione OPEN, una periferica che si sappia per certo trattare le stringhe IOCTL, non si può capire come ottenerle né interpretarle senza avere dettagliate informazioni sulle caratteristiche e sul funzionamento della stessa.

Il seguente programma, generico in quanto non dedicato ad un preciso device driver, indica le modalità d'uso della funzione IOCTL$

```
CLS
OPEN … AS #1               ' Apertura del canale con il device driver
IOCTL #, "…"               ' Stringa di comando al device driver
PRINT IOCTL$(1)            ' Richiesta di una stringa di stato
CLOSE
END
```

*** LBOUND**

Questa funzione è usata per determinare il **valore minimo** dell'indice di una determinata dimensione di un array. Nella sua sintassi

LBOUND (array [,dimensione])

il primo parametro è il nome dell'array mentre il secondo, che è opzionale, è la dimensione di cui si vuole l'indice minimo; in mancanza di tale argomento, viene ritornato dalla funzione l'indice minimo relativo alla prima dimensione dell'array. Non è possibile indicare in 'dimensione' un valore che eccede il numero di dimensioni dell'array specificato.
L'uso di tale funzione, insieme alla **UBOUND**, funzione gemella che serve per determinare il **valore massimo** dell'indice di una determinata dimensione (la sintassi è la stessa), è molto diffuso nelle Functions e nelle Subs che non conoscono a priori i valori minimi e massimi degli indici degli arrays che sono passati come argomento. Nell'esempio seguente infatti, si vede come le due funzioni possano essere usate all'interno di una Sub generica che serve ad azzerare un vettore numerico, qualsiasi limiti esso abbia

```
DECLARE SUB ZEROVEC(X!())
CLS
DIM W(10 TO 300)
ZEROVEC W( )
END
SUB ZEROVEC(X())
    FOR Y=LBOUND(X) TO UBOUND(X)
            X(Y)=0
    NEXT Y
END SUB
```

*** LCASE$**

Questa funzione ritorna una stringa uguale a quella passata come argomento, tranne per il fatto che le lettere dell'alfabeto sono tutte trasformate in minuscolo. È simile alla funzione gemella **UCASE$** che opera in maniera contraria; essa serve cioè, a trasformare tutte le lettere minuscole dell'argomento, in maiuscole. Le due funzioni sono utili quando, durante una ricerca di dati in un file, bisogni comparare una stringa ricercata con il contenuto dei records; se si i dati letti da disco che la stringa ricercata vengono, prima della comparazione, trasformati in maiuscolo (o minuscolo, senza alcuna differenza), si può essere sicuri che la ricerca stessa opererà in modo corretto (vedere anche la funzione INSTR).
Come esempio del funzionamento delle funzioni LCASE$ ed UCASE$, seguire il funzionamento del seguente programma

```
CLS
X$=""
FOR X=65 TO 90
        X$=X$+CHR$(X)
NEXT X
X$=X$+"_"
FOR X=97 TO 122
        X$=X$+CHR$(X)
NEXT X
PRINT "Stringa di prova    : "; X$
PRINT TAB(11); X$
PRINT
PRINT
PRINT TAB(29); "Stringa dopo LCASE$"
PRINT TAB(11); LCASE$(X$)
PRINT
PRINT TAB(29); "Stringa dopo UCASE$"
PRINT TAB(11); UCASE$(X$)
END
```

il cui risultato sullo schermo è il seguente

```
                    Stringa di prova
ABCDEFGHIJKLMNOPQRSTUVWXYZ_abcdefghijklmnopqrstuvwxyz
```

Stringa dopo LCASE$
abcdefghijklmnopqrstuvwxyz_abcdefghijklmnopqrstuvwxyz

Stringa dopo UCASE$
ABCDEFGHIJKLMNOPQRSTUVWXYZ_ABCDEFGHIJKLMNOPQRSTUVWXYZ

*** LEFT$**

Questa funzione ritorna una stringa formata degli n caratteri, a partire dalla sinistra, della stringa usata come argomento. La sua sintassi è

LEFT$(estrstringa, n)

in cui esprstringa può essere una costante, una variabile o un'espressione stringa. Il secondo parametro rappresenta invece il numero di caratteri, a partire da sinistra, che devono essere prelevati da esprstringa e ritornati dalla funzione.

Il funzionamento di LEFT$ è molto simile a quello di RIGHT$ ed è a quest'ultima funzione che si rimanda per l'esempio.

*** LEN**

È una funzione numerica che ha come argomento una stringa. La sua sintassi è dunque

LEN(esprstringa)

Se esprstringa è una stringa nulla, la funzione ritorna il valore zero, altrimenti ritorna il numero di caratteri compresi in esprstringa. Porre attenzione al fatto che tutti i caratteri, compresi gli spazi, se appartenenti ad esprstringa, sono ritenuti validi e contati. Ad esempio, la seguente linea di programma

PRINT LEN("Questa e' una prova")

visualizzerà il numero 19 perché questo è il numero di caratteri inclusi nella stringa (le virgolette non fanno parte di quest'ultima ma la delimitano).

*** LOC**

La sintassi di questa funzione di I/O è la seguente

LOC(nfile)

in cui l'unico argomento è il numero di un file già aperto ed in uso. La funzione ritorna un valore numerico che può essere interpretato diversamente a seconda del tipo di file a cui fa riferimento la LOC. Più precisamente

se il file è sequenziale la funzione ritorna il valore della posizione dell'ultimo carattere letto o scritto diviso per 128 senza decimali;

se il file è random la funzione ritorna il numero dell'ultimo record letto con l'istruzione GET o scritto con l'istruzione PUT;

se il file è binario la funzione ritorna il valore della posizione dell’ultimo carattere letto o scritto;

se il file aperto fa riferimento al device COM la funzione ritorna il numero dei caratteri in coda nel buffer di ricezione.

La funzione LOC non può essere usata per i device SCRN:, KYBD: e LPTn:

Ecco un esempio di come è possibile sfruttare le funzioni LOC e LOF (per ricavare la lunghezza di un file) per realizzare una funzione che divida in due parti un file; la Sub costruita (SplitFile) accetta i seguenti parametri

- nome del file da spezzare
- nome del file che costituirà la prima parte
- nome del file che costituirà la seconda parte
- lunghezza del file che costituirà la prima parte in percentuale rispetto alla lunghezza del file da spezzare

```
DECLARE SUB SplitFile(Fa0$, Fa1$, Fa2$, Perc1%)
CLS
SplitFile “C:\AUTOEXEC.BAT”, “A.SP1”, “B.SP2”, 30
END
SUB SplitFile(Fa0$, Fa1$, Fa2$, Perc1%)
        CH$=” ”
        F0 = FREEFILE
        OPEN Fa0$ FOR BINARY AS F0
        F1 = FREEFILE
        OPEN Fa1$ FOR BINARY AS F1
        F2 = FREEFILE
        OPEN Fa2$ FOR BINARY AS F2
        L0 = LOF(F0)
        L1 = CINT(L0/100*Perc1%)
        L2 = L0 - L1
        LOCATE 22, 22
        PRINT “Lettura di “; Fa0$; “ (1^ parte)”
        LOCATE 23, 30
        PRINT “Scrittura di “; Fa1$;
        LOCATE 24, 15
        PRINT STRING$(50, 46);
        FOR By=1 TO L1
                GET F0,,CH$
                PUT F1,,CH$
                Xp=50*LOC(F1)/L1
                LOCATE 24, 15
                PRINT STRING$(Xp, 178);
                LOCATE 24, 72
                PRINT USING “(###%)”; Xp*2;
```

```
		NEXT By
		CLS
		LOCATE 22, 22
		PRINT “Lettura di “; Fa0$; “ (2^ parte)”
		LOCATE 23, 30
		PRINT “Scrittura di “; Fa2$;
		LOCATE 24, 15
		PRINT STRING$(50, 46);
		FOR By=1 TO L2
			GET F0,,CH$
			PUT F2,,CH$
			Xp=50*LOC(F2)/L2
			LOCATE 24, 15
			PRINT STRING$(Xp, 178);
			LOCATE 24, 72
			PRINT USING “(###%)”; Xp*2;
		NEXT By
		CLOSE F0, F1, F2
	END SUB
```

*** LOF**

Questa funzione ritorna la lunghezza in bytes di un file aperto in precedenza e in uso. Il numero del file a cui si fa riferimento, va specificato come argomento; la sintassi della funzione è

LOF(n)

Se nell’istruzione di apertura del file si fa riferimento al device COM, il valore ritornato dalla funzione rappresenta il numero di bytes liberi nel buffer di ricezione. Con tutte le altre periferiche standard, non è possibile usare questa funzione.

Per quanto riguarda i file random, l’uso di questa funzione si rende necessario quando si deve ricavare il numero dei records presenti nel file, con la formula

NumRecs = LunghFile / LunghRec

come nel seguente esempio

```
	NOME$=”PROVA.DAT”
	LREC=200
	CLS
	OPEN NOME$ FOR RANDOM AS 1 LEN=LREC
	FIELD #1, 100 AS P1$, 100 AS P2$
	FOR X=1 TO 10
		PUT #1
		NR=LOF(1)/LREC
		PRINT “Nel file “; NOME$; “ ci sono “; NR; “ records.”
	NEXT X
	CLOSE
```

```
END
```

*** LOG**

È una delle funzioni matematiche del QB, e serve a calcolare il logaritmo in base naturale, del numero usato come argomento. Quest'ultimo può essere solamente maggiore di zero, altrimenti viene visualizzato un errore.

La precisione del risultato è quella adottata per l'argomento.
Per ottenere i valori dei logaritmi in qualsiasi base, basta adottare la seguente formula

L=LOG(N)/LOG(B)

in cui B è la base prescelta, N l'argomento ed L il valore del logaritmo in base B di N.

Ad esempio, per calcolare il logaritmo in base 10 dei numeri 10, 100, 1000 e 10000, bastano le seguenti linee di programma

```
DEF FNLOG10(N) = LOG(N) / LOG(10)
CLS
FOR NU=10 TO 10000
      PRINT NU, FNLOG10(N)
      NU=NU*10-1
NEXT NU
END
```

Per quanto riguarda la quinta linea di questo programma, ricordare che, ad ogni passo del ciclo FOR..NEXT, la variabile viene aumentata di 1 e che per ottenere i valori 10, 100, 1000 e 10000 bisogna moltiplicare il precedente per 10.

*** LPOS**

Questa funzione ritorna la posizione corrente della testina di stampa della stampante selezionata come argomento. La posizione, in effetti, fa riferimento al buffer interno della stampante e potrebbe quindi, differire da quella fisicamente assunta dalla testina stessa.
La sintassi della funzione è

LPOS(nstamp)

in cui 'nstamp' è un valore compreso tra 0 e 3 il cui significato è il seguente

0 e **1** riferimento alla stampante LPT1:
2 riferimento alla stampante LPT2:
3 riferimento alla stampante LPT3:

Nel seguente esempio, si possono notare a video le posizioni di stampa dei dati, ricavati tramite la funzione LPOS

```
CLS
PRINT
```

```
H$="1234567890"
RH$=""
FOR X=1 TO 8
       RH$=RH$+H$
NEXT X
PRINT RH$
LPRINT
FOR X=1 TO 20
       PRINT TAB(LPOS(1)); X;
       LPRINT X;
NEXT X
END
```

*** LTRIM$**

Questa funzione può essere usata ogni volta che è necessario rimuovere gli spazi presenti all'inizio di una stringa. Con la funzione gemella, **RTRIM$**, che rimuove gli spazi finali, costituisce un veloce metodo per rimediare al fatto che i dati letti dall'interno dei campi in un file random siano sempre completati da spazi finali; eliminando tali spazi ed, eventualmente, anche quelli iniziali posti per sbaglio dall'utente, si ottengono le premesse per la realizzazione di routines di ricerca dei dati più efficienti (vedere funzioni LCASE$, UCASE$, INSTR). Nell'esempio che segue, notare la differenza delle lunghezze dei dati letti da un file random (gli asterischi sono stati posti appositamente per evidenziare la presenza degli spazi iniziali e finali)

```
CLS
OPEN "PRO" FOR RANDOM AS 1 LEN=30
FIELD #1, 30 AS Nomin$
FOR N=1 TO 5
    READ N$
    LSET Nomin$=N$
    PUT #1, N
NEXT N
FOR N=1 TO 5
    GET #1, N
    PRINT "*"; Nomin$; "*", "*"; LTRIM$(RTRIM$(Nomin$)); "*"
NEXT N
CLOSE
END
DATA " Rossi Paolo", "   Verdi Enrico"
DATA " Gialli Piero", "Bianchi Mauro", "   Neri Luigi"
```

*** MID$**

Questa funzione ritorna una stringa, estratta dall'interno di un'altra usata come argomento. La sua sintassi è la seguente

MID$(esprstringa, n, [,n])

in cui esprstringa è una costante, una variabile o espressione stringa e i due valori numerici, di cui uno opzionale, indicano

m è la posizione del carattere di esprstringa, a partire da sinistra, ad iniziare dal quale bisogna estrarre la stringa voluta; il suo valore può andare da 1 a 32767;

n è il numero di caratteri da cui sarà composta la stringa ritornata, a partire dal carattere m-esimo; se questo valore manca, la stringa ritornata è composta da tutti i caratteri di esprstringa a partire dal carattere m-esimo fino alla sua fine.

Per la funzione che MID$ esplica, molto simile a quella di LEFT$ e RIGHT$, si rimanda per l'esempio a quello usato in quest'ultima.

*** MKD$**
Vedere CVD

*** MKDMBF$**
Vedere CVDMBF

*** MKI$**
Vedere CVI

*** MKL$**
Vedere CVL

*** MKS$**
Vedere CVS

*** MKSMBF$**
Vedere CVSMBF

*** OCT$**
La sintassi di questa funzione è la seguente

OCT$(n)

in cui n è l'argomento numerico che verrà convertito in notazione ottale. La stringa ritornata dalla funzione è di lunghezza variabile e il valore n è automaticamente convertito in intero ed arrotondato se è decimale; vengono gestiti gli interi lunghi (interi con segno a 32 bits). Il seguente esempio dimostra come è possibile convertire valori numerici dal sistema decimale all'ottale

```
CLS
FOR X&=0 TO 20
   PRINT X&, OCT(X&)
NEXT X&
END
```

*** PEEK**

Questa funzione è usata per ricavare il contenuto di una specifica locazione di memoria del sistema, all'interno di un determinato segmento. Quest'ultimo può essere predisposto tramite una istruzione DEF SEG; la funzione PEEK ha la seguente sintassi

PEEK(offset)

in cui il primo parametro rappresenta l'indirizzo della cella di memoria del segmento corrente da leggere; tale indirizzo è un valore compreso tra 0 e 65535 (0000…FFFF esadecimale), mentre il valore restituito è compreso tra 0 e 255 (00…FF esadecimale). L'istruzione **POKE** è complementare alla funzione PEEK in quanto serve a scrivere in una determinata cella di memoria, se possibile.

La prima parte di RAM del sistema è usata dal firmware e dal software di base (quello contenuto nelle ROM del BIOS e quello facente parte del DOS) e dalle celle di questa zona è possibile, conoscendo i giusti indirizzi, ricavare utili informazioni sullo stato del computer. Ad esempio, nel segmento 40h (esadecimale) esistono diverse locazioni di memoria utilizzate dalla ROM del BIOS e dal DOS ed, in particolare, le celle 17h e 18h (esadecimale) che sono continuamente aggiornate sull'uso dei tasti Shift, Ctrl, Alt, Num Lock, Caps Lock, Scroll Lock, Sys Req; il seguente esempio dimostra il modo con cui è possibile ricavare tali informazioni tramite la funzione PEEK (questo programma è interrompibile premendo il tasto Alt+SysReq)

```
CLS
DEF SEG=&H40
DO
        A=PEEK(&H17)
        B=PEEK(&H18)
        PRINT HEX$(A); " "; HEX$(B)
        IF B=6 THEN
                EXIT DO
        END IF
LOOP
DEF SEG
END
```

*** PEN**

È la funzione usata per controllare la penna ottica, quando questa periferica è collegata al sistema. La sua sintassi

PEN(funz)

prevede un argomento numerico compreso tra 0 e 9 che assume il significato della seguente tabella

0 restituisce -1 se la penna è stata appoggiata al video dopo l'ultima chiamata di PEN(0); altrimenti restituisce 0;

1 restituisce la coordinata x dell'ultimo punto in cui la penna è stata appoggiata;

2 restituisce la coordinata y dell'ultimo punto in cui la penna è stata appoggiata;

3 restituisce -1 se l'interruttore della penna è abbassato; altrimenti restituisce 0;

4 restituisce l'ultima coordinata x conosciuta;

5 restituisce l'ultima coordinata y conosciuta;

6 restituisce il numero della riga in cui la penna è sata appoggiata l'ultima volta;

7 restituisce il numero della colonna in cui la penna è sata appoggiata l'ultima volta;

8 restituisce l'ultimo numero di riga conosciuto;

9 restituisce l'ultimo numero di colonna conosciuto.

La funzione PEN deve essere preceduta dall'istruzione PEN ON perché lavori correttamente; è inoltre necessario non abilitare il driver per il mouse in quanto i due dispositivi risultano, normalmente, incompatibili.

Nel seguente esempio, vengono mostrate a video le coordinate della penna

```
CLS
PEN ON
DO
    LOCATE 10, 10
    PRINT USING "(x) ###  (y) ###"; PEN(3); PEN(4)
LOOP WHILE INKEY$=""
PEN OFF
END
```

*** PLAY**

La funzione PLAY restituisce il numero di note presenti nel buffer di esecuzione quando si è in modalità background. La sintassi della funzione è la seguente

PLAY(n)

in cui n è un argomento numerico qualsiasi che non influenza il valore restituito dalla funzione (argomento dummy).

Se il modo di funzionamento dell'istruzione PLAY eseguita in precedenza è il foreground, la funzione PLAY restituisce il valore zero.
Il seguente esempio dimostra come sia possibile, con questa funzione, conoscere il numero delle note ancora da suonare se si agisce in background (comando MB dell'istruzione PLAY)

```
CLS
PLAY "MBL8CDEFGCDEFGCDEFGCDEFG"
```

```
LOCATE 9, 8
PRINT “Modalita’ background …”
DO WHILE PLAY(0)>0
        LOCATE 10, 10
        PRINT “Note da suonare : “; PLAY(0)
LOOP
END
```

*** PMAP**

La funzione PMAP è usata da programmi che sfruttano la grafica, in quanto restituisce delle informazioni riguardanti le coordinate dei punti visualizzabili in modalità ad alta risoluzione. La sua sintassi è

PMAP(cooespr, funz)

prevede due argomenti; il primo è il valore della coordinata del punto che vuole trattare mentre, il secondo, è un numero compreso tra 0 e 3, la cui funzione è esposta nella seguente tabella

0 la funzione restituisce la coordinata x fisica relativa al valore logico espresso come primo argomento;

1 la funzione restituisce la coordinata y fisica relativa al valore logico espresso come primo argomento;

2 la funzione restituisce la coordinata x logica relativa al valore della coordinata fisica espresso nel primo argomento;

3 la funzione restituisce la coordinata y logica relativa al valore della coordinata fisica espresso nel primo argomento.

Le coordinate logiche di un punto sono quelle determinate dall’istruzione WINDOW mentre le fisiche, dall’istruzione VIEW; in mancanza di queste istruzioni le coordinate fisiche e logiche corrispondono.

Se, ad esempio, si eseguisse il seguente programma

```
CLS
SCREEN 2
WINDOW SCREEN (0, 0)-(1, 1)
PRINT PMAP(1, 0), PMAP(1, 1)
END
```

verrebbe visualizzata la coordinata x (639) ed y (199), valori massimi nella risoluzione scelta dato che sono state convertire le coordinate logiche massime definite con l’istruzione WINDOW.

Il programma di esempio che segue usa l’istruzione WINDOW per definire le dimensioni logiche del video e, con l’istruzione PSET, visualizza la curva delle funzioni seno e coseno; la funzione

PMAP è utilizzata per fare in modo di rilevare i punti in cui le due curve di toccano e visualizzare le coordinate x logiche

```
CONST PI=3.14159
CLS
SCREEN 2
WINDOW (-3*PI, -1)-(3*PI, 1)
Y=1
LINE (-3*PI, 0)-(3*PI, 1)
LINE (0, -1)-(0, 1)
FOR X=-3*PI TO 3*PI STEP .01
        LOCATE Y, 1
        PRINT USING "+#.###"; X
        IF PMAP(SIN(X), 1)=PMAP(COS(X), 1) THEN
                Y=Y+1
        END IF
        PSET (X, SIN(X))
        PSET (X, COS(X))
NEXT X
LOCATE Y, 1
PRINT "      ";
A$=INPUT$(1)
END
```

*** POINT**

È una funzione di tipo grafico con due sintassi. La prima

POINT (xcoo, ycoo)

si usa quando si deve ottenere il colore del punto che ha coordinate xcoo e ycoo (coordinate fisiche se non è stata usata l'istruzione WINDOW, altrimenti logiche).
La seconda sintassi

POINT (funz)

prevede un solo argomento numerico che può assumere i valori compresi tra 0 e 3; in questo modo la funzione restituisce diversi valori secondo la tabella seguente

0 la funzione restituisce il valore della coordinata x fisica attuale;

1 la funzione restituisce il valore della coordinata y fisica attuale;

2 la funzione restituisce il valore della coordinata x logica attuale (se è stata usata la WINDOW);

3 la funzione restituisce il valore della coordinata y logica attuale (se è stata usata la WINDOW).

Il seguente programma di esempio, mostra un punto in grafica monocromatica CGA, e le sue coordinate fisiche e logiche; si è definito lo schermo con le coordinate logiche -pigreco per la x e -1..+1 per la y; il punto può essere mosso con i tasti del cursore ad una bassa velocità mentre, se si preme anche lo Shift, ad una velocità più elevata; il tasto Esc fa terminare il programma

```
DECLARE SUB POO( )
CLS
SCREEN 2
WINDOW (-3.14, -1)-(3.14, 1)
DEF SEG=&H40
X=POINT(2)
Y=POINT(3)
PCOO
DO
  PSET (X, Y)
  A$=INKEY$
  SHF=PEEK(&H17) AND 2
  SELECT CASE LEN(A$)
          CASE 1
                  A=ASC(A$)
          CASE 2
                  A=ASC(MID$(A$, 2))
          CASE ELSE
                  A=0
  END SELECT
  IF A>0 THEN
          PRESET (X, Y)
          IF SHF=2 THEN
                  STPX=.5
                  STPY=.1
          ELSE
                  STPX=.01
                  STPY=.01
          END IF
          SELECT CASE A
                  CASE 72
                          Y=Y+STPY
                  CASE 75
                          X=X-STPX
                  CASE 77
                          X=X+STPX
                  CASE 80
                          Y=Y-STPY
                  CASE 27
                          SOUND 500, 2
                          EXIT DO
                  CASE ELSE
```

```
            END SELECT
            IF X<-3.14 THEN X=-3.14
            IF Y<-1 THEN Y=-1
            IF X>3.14 THEN X=3.14
            IF Y>1 THEN Y=1
            PCOO
  END IF
LOOP
DEF SEG
WINDOW
SCREEN 0
END
SUB PCOO
  LOCATE 1, 10
  PRINT USING "Coo Fis. ### (x) ### (y)"; POINT(0); POINT(1)
  LOCATE 2, 10
  PRINT USING "Coo Log. +### (x) +### (y)"; POINT(2); POINT(3)
END SUB
```

*** RIGHT$**

È una funzione di tipo stringa; essa ritorna gli n caratteri alla destra della stringa usata come argomento. La sua sintassi è la seguente

RIGHT$(esprstringa, n)

in cui esprstringa può essere una costante, una variabile o una espressione stringa ed n un valore positivo che indica il numero dei caratteri di esprstringa, a partire da destra, che la funzione deve ritornare. Se n è zero, viene ritornata una stringa nulla, mentre se n è maggiore o uguale alla lunghezza di esprstringa, viene ritornata tutta esprstringa. Questa funzione è molto simile alla LEFT$.

Ad esempio, essa può essere utilizzata per cambiare il formato di una data contenuta in una stringa, da quello GG-MM-AAAA a quello AAAAMMGG (utile per essere memorizzato) e viceversa

```
CLS
D$="01-05-2000"
DM$=RIGHT$(D$, 4)+MID$(D$,4,2)+LEFT$(D$,2)
PRINT DM$
DV$= RIGHT$(D$, 4)+MID$(D$,4,2)+LEFT$(D$,2)
PRINT DV$
END
```

*** RND**

Questa funzione ritorna un valore numerico a caso, in singola precisione, compreso tra 0 e 1. La sua sintassi è la seguente

RND [**(n)**]

in cui l’unico parametro è opzionale. Tale valore determina la generazione del numero casuale nelle modalità previste dalla seguente tabella

n < 0 la funzione ritorna sempre lo stesso numero per ogni valore di n. L’istruzione RANDOMIZE non influenza la funzione RND usata così;

n = 0 la funzione ritorna l’ultimo numero casuale generato in precedenza; se nessun numero era stato generato prima, ritorna il primo numero casuale della sequenza;

n > 0 la funzione ritorna il prossimo numero casuale della sequenza. L’istruzione RANDOMIZE serve a definire la sequenza da usare.

Se l’argomento non dovesse essere specificato, la funzione si comporterebbe come nell’ultimo caso (n > 0).

Per ottenere un numero casuale intero, compreso tra due valori, minimo e massimo, basta usare la seguente formula

INT((massimo-minimo+1)*RND+minimo)

Nel seguente esempio, si può notare l’utilizzo di RND(0) per ricavare l’ultimo numero casuale generato (premendo il tasto P)

```
RANDOMIZE TIMER
CLS
DO
	PRINT “Adesso e’ uscita “;
	IF RND>=.5 THEN
		PRINT “testa”
	ELSE
		PRINT “croce”
	END IF
	A$=INPUT$(1)
	IF UCASE$(A$)=”P” THEN
		PRINT TAB(10); “Nel lancio precedente e’ uscita “;
		IF RND(0)>=.5 THEN
			PRINT “testa”
		ELSE
			PRINT “croce”
		END IF
	END IF
LOOP WHILE A$<>CHR$(27)
END
```

*** RTRIM$**

Vedere LTRIM$

*** SADD**

Questa funzione è molto usata nei programmi che fanno uso di parti scritte in Assembler, C o altri linguaggi. Essa ritorna l'indirizzo del **contenuto di una espressione alfanumerica** all'interno del segmento appropriato. È usata insieme alla funzione VARSEG per determinare l'indirizzo completo del contenuto di una variabile stringa.
Il Quick Basic memorizza per ogni stringa, 4 bytes, (descrittore della stringa) in cui conserva

primi 2 bytes parte bassa ed alta della lunghezza attuale della stringa (può essere compresa tra 0 e 32767):

seguenti 2 bytes parte bassa ed alta dell'indirizzo del contenuto della stringa.

Il funzionamento della SADD può essere quindi simulato con la funzione VARPTR; infatti, il seguente programma

```
CLS
A$="Ciao"
S=SADD(A$)
PRINT S
END
```

e quest'altro

```
CLS
A$="Ciao"
S=PEEK(VARPTR(A$)+2)+PEEK(VARPTR(A$)+3)*256
PRINT S
END
```

sono equivalenti.

Quindi, mentre la funzione VARPTR applicata ad una stringa, ritorna l'indirizzo **del descrittore** suddetto, la funzione SADD ritorna il valore degli ultimi due bytes del descrittore, contenenti, appunto, l'indirizzo del **contenuto** della stringa.

Il seguente esempio mostra la differenza suddetta

```
DEFINT A-Z
CLS
A$="Quick Basic"
S=SADD(A$)
V=VARPTR(A$)
PRINT "All'indirizzo "; HEX$(V); " fornito dalla VARPTR(A$)"
PRINT TAB(12); "c'e' il descrittore di A$ che contiene … ";
FOR X=V TO V+3
	PRINT HEX$(PEEK(X)); " ";
```

```
NEXT X
PRINT
PRINT
PRINT “I primi due bytes, invertiti, “;
PRINT “sono la lunghezza di A$ -> “;
PRINT HEX$(PEEK(V+1)); HEX$(PEEK(V))
PRINT
PRINT “Gli ultimi due bytes del descrittore “;
PRINT “formano l’indirizzo -> “;
PRINT HEX$(PEEK(V+3)); HEX$(PEEK(V+2))
PRINT TAB(12); “che è anche il valore “;
PRINT “restituito da SADD(A$) -> “; HEX$(S)
PRINT
PRINT “All’indirizzo suddetto, “;
PRINT “per la lunghezza di A$ si trova … “
PRINT
FOR X= S TO S+LEN(A$)-1
        PRINT HEX$(PEEK(X)); “ (“; CHR$(PEEK(X)); “) “;
NEXT X
PRINT
PRINT
PRINT “… il contenuto della stringa.”
END
```

Nella libreria BPLUS, che è fatta anche con altri linguaggi (Assembler, C), la funzione SADD è molto usata.

*** SCREEN**

La funzione SCREEN (da non confondersi con l’omonima istruzione), ha la seguente sintassi

SCREEN (riga, colonna [**, flcol**]**)**

in cui riga e colonna costituiscono le coordinate, in modo testo, di un carattere sul video. Se l’ultimo argomento non viene specificato (oppure è uguale a zero), la funzione ritorna il **codice ASCII** del carattere che è sul video alle coordinate indicate; se l’argomento flcol viene indicato uguale a 1, la funzione ritorna il valore del **colore** del suddetto carattere.

Il seguente esempio mostra il funzionamento di SCREEN, visualizzando i valori 81 e 15 per la lettera Q in bianco intenso, e 66 e 7 per la lettera B bianca

```
CLS
LOCATE 10, 10
COLOR 15
PRINT “Q”;
COLOR 7
PRINT “B”
LOCATE 20, 1
PRINT SCREEN(10, 10, 0); SCREEN(10, 10, 1)
```

```
PRINT SCREEN(10, 11, 0); SCREEN(10, 11, 1)
END
```

*** SEEK**

La funzione SEEK, la cui sintassi e’

SEEK(numfile)

restituisce, per i file di tipo RANDOM, il numero del prossimo record da scrivere o leggere. Per gli altri tipi di file, viene restituito il numero del byte corrente che può essere scritto o letto. L’argomento della funzione è uguale al numero usato nell’istruzione OPEN relativa.

Per le periferiche che non supportano la funzione SEEK (SCRN:, CONS:, KYBD:, COMn: e LPTn:) essa ritorna sempre il valore 0.

Il seguente programma visualizza una percentuale a scelta del file AUTOEXEC.BAT, controllando la parte visualizzata con la funzione SEEK

```
CLS
OPEN “C:\AUTOEXEC.BAT” FOR INPUT AS 1
INPUT “Percentuale : “, PERC
DO WHILE SEEK(1)<PERC*LOF(1)/100
        K$=INPUT$(1, 1)
        IF ASC(K$)<>&HA THEN
                PRINT K$;
        END IF
LOOP
CLOSE
END
```

*** SETMEM**

Questa funzione è usata per modificare la dimensione della memoria FAR allocata da Quick Basic. La sua sintassi è

SETMEM(bytes)

in cui bytes è il numero di bytes che si vogliono togliere (se bytes <0) o aggiungere (bytes >0) alla zona di memoria FAR.
All’inizio di un programma, la zona di memoria FAR è allocata completamente rendendo inutile la funzione SETMEM usata per aggiungere memoria. Dopo essere stata usata, la funzione ritorna sempre il valore della dimensione attuale della memoria FAR; per conoscere tale valore senza modificarlo, si può richiamare la funzione con argomento 0.
Questa funzione è indispensabile quando si deve chiamare una procedura scritta in un altro linguaggio, che alloca memoria FAR autonomamente; prima di chiamare tale procedura si deve procedere ad un’adeguata diminuzione della memoria FAR allocata da QB e prima di tornare al programma in Basic, la procedura scritta in un altro linguaggio, deve liberare la memoria allocata alla sua partenza. La sequenza corretta, ad esempio con programmi scritti in C, è la seguente

1) liberazione di memoria FAR con la funzione SETMEM con argomento negativo;
2) chiamata della procedura scritta in C;
3) chiamata della funzione malloc() del C per allocare la memoria FAR alla procedura;
4) uso della mamoria allocata da parte della procedura;
5) chiamata della funzione free() del C per liberare la memoria FAR utilizzata dalla procedura;
6) ritorno al programm in QB;
7) chiamata della funzione SETMEM con argomento positivo per riallocare la memoria FAR.

Nel seguente esempio, si eseguono tutte le operazioni suddette in Quick Basic, mentre si ipotizza l'esecuzione di un programma in C

```
CLS
PRINT "Dimensione della zona FAR : "; SETMEM(0); "bytes"
PRINT
PRINT "Tolti 100000 bytes per allocarli con C."
PRINT "Dimensione della zona FAR : "; SETMEM(-100000); "bytes"
PRINT
PRINT "Esecuzione del programma C."
PRINT
PRINT "Aggiunti 100000 bytes ora disponibili"
PRINT "Dimensione della zona FAR : "; SETMEM(100000); "bytes"
END
```

*** SGN**

La funzione SGN ritorna un valore diverso a seconda del segno del suo argomento numerico. La sua sintassi è

SGN(espnum)

in cui espnum è l'espressione numerica di cui si deve valutare il segno; la funzione ritorna i seguenti valori in dipendenza di tale segno

-1 se espnum è negativa;
0 se espnum è zero;
1 se espnum è positiva

Nel seguente programma di esempio, si può notare l'uso della funzione SGN per valutare la possibilità di calcolare o meno una funzione

```
CONST Negativo=-1, Positivo=1, Zero=0
DO
    CLS
    INPUT "Valore dell'argomento : ", N
    SELECT CASE SGN(N)
        CASE Negativo
```

```
                PRINT “La Radice Quadrata di “; N; “ non è definita.”
                PRINT “Il Logaritmo di “; N; “ non è definito.”
            CASE Zero
                PRINT “La Radice Quadrata di 0 e’ 0.”
                PRINT “Il Logaritmo di 0 non è definito.”
            CASE Positivo
                PRINT “La Radice Quadrata di “; N; “e’”; SQR(N)
                PRINT “Il Logaritmo di “; N; “e’”; LOG(N)
        END SELECT
    LOOP WHILE UCASE$(INPUT$(1)) <> “F”
    END
```

*** SIN**

La funzione SIN restituisce il valore del seno dato come argomento l’angolo espresso in radianti. È una delle funzioni trigonometriche del QB, la cui precisione del valore ritornato dipende dalla precisione dell’argomento.

Per un esempio sull’uso della funzione SIN, vedere l’esempio riguardante la funzione PMAP.

*** SPACE$**

È una funzione che ritorna una stringa formata da n spazi, in cui n può essere compreso tra 0 (stringa nulla) e 32767. La sua sintassi è

SPACE$(n)

Essa può essere usata, ad esempio, per cancellare parti di schermo, come nel seguente esempio

```
‘ (Cancella da riga 3 a riga 8, da colonna 10 a 40)
FOR R=3 TO 8
    LOCATE R, 10
    PRINT SPACE$(30);
NEXT R
```

*** SPC**

Questa funzione può essere usata solamente insieme alle istruzioni PRINT ed LPRINT (a differenza della SPACE$ che può essere usata anche per assegnare spazi ad una variabile stringa); la sua sintassi

SPC(n);

indica che ha un solo argomento; questo è un numero compreso tra 0 e 32767 ed indica il numero degli spazi che devono essere visualizzati o stampati. Il simbolo ; alla fine della funzione è **obbligatorio** così che l’uso di tale funzione alla fine di una istruzione PRINT o LPRINT non fa tornare a capo il cursore o la testina di stampa. Se dovesse invece, essere necessario il ritorno a capo, è d’obbligo usare la funzione SPACE$.
Nei prossimi due esempi, si può notare quest’ultima caratteristica della funzione SPC e come si può ovviare con la SPACE$

```
' (Programma con SPC)
CLS
LOCATE 10, 7
PRINT STRING$(30, "!")
LOCATE 10, 7
PRINT "Quick Basic"; SPC(10);
PRINT "Programmazione Avanzata"
END

' (Programma con SPACE$)
CLS
LOCATE 10, 7
PRINT STRING$(30, "!")
LOCATE 10, 7
PRINT "Quick Basic"; SPACE$(10);
PRINT "Programmazione Avanzata"
END
```

*** SQR**

Questa funzione ritorna il valore della radice quadrata del suo argomento che, naturalmente, deve essere compreso tra zero e l'infinito-macchina.

Il seguente programma, visualizza una tabella con i valori delle radici quadrate dei primi 10 numeri interi

```
CLS
FOR N=1 TO 10
    PRINT N, SQR(N)
NEXT N
```

La precisione del risultato è quella dell'argomento usato.

*** STICK**

Questa funzione è usata per controllare la posizione di due joystick (periferiche di input usate, per lo più, per i videogiochi). La sua sintassi è

STICK(funz)

in cui funz è un numero che può assumere i seguenti valori della tabella seguente in cui sono specificate anche le operazioni conseguenti

0 la funzione restituisce il valore della coordinata x del joystick A;

1 la funzione restituisce il valore della coordinata y del joystick A; usare questo valore solo dopo avere eseguito una volta la funzione con argomento 0;

2 la funzione restituisce il valore della coordinata x del joystick B; usare questo valore solo dopo avere eseguito una volta la funzione con argomento 0;

3 la funzione restituisce il valore della coordinata y del joystick B; usare questo valore solo dopo avere eseguito una volta la funzione con argomento 0.

Dato che la funzione con argomento 0, non solo restituisce il valore della coordinata x del joystick A, ma memorizza internamente le altre coordinate che possono essere lette di seguito, questa **deve essere eseguita prima** delle altre. Il seguente esempio, mostra come è possibile costruire una funzione che restituisca le coordinate di uno dei due joystick e le adatti ad un dato intervallo considerando che il valore di quest'ultime è sempre compreso tra 1 e 200

```
DECLARE FUNCTION JOYSTICK!(JOY%, COO%, MIN!, MAX!)
CLS
SCREEN 2
XA=POINT(0)
YA=POINT(1)
PSET(XA, YA)
XB=POINT(0)
YB=POINT(1)
PSET(XB, YB)
DO
    NXA=JOYSTICK(0, 0, 0, 639)
    NYA=JOYSTICK(0, 1, 0, 199)
    IF NXA<>XA OR NYA<>YA THEN
        PRESET (XA, YA)
        XA=NXA
        YA=NYA
        PSET(XA, YA)
    END IF
    NXB=JOYSTICK(0, 0, 0, 639)
    NYB=JOYSTICK(0, 1, 0, 199)
    IF NXB<>XB OR NYB<>YB THEN
        PRESET (XB, YB)
        XB=NXB
        YB=NYB
        PSET(XB, YB)
    END IF
LOOP WHILE INKEY$=""
SCREEN 0
END
' (Funzione Joystick)
' Parametro JOY%          → 0=Joystick A, 1=Joystick B
' Parametro COO%          → 0=Coord. x, 1=Coord. y
' Parametri MIN! e MAX!   → Intervallo di valori
FUNCTION JOYSTICK(JOY%, COO%, MIN, MAX) STATIC
    X1=STICK(0)
    Y1=STICK(1)
    X2=STICK(2)
    Y2=STICK(3)
```

```
    IF COO%=0 THEN
        IF JOY%=0 THEN
            V=X1
        ELSE
            V=X2
        END IF
    ELSE
        IF JOY%=0 THEN
            V=Y1
        ELSE
            V=Y2
        END IF
    END IF
    JOYSTICK=CINT((V/200) * (MAX-MIN) + MIN)
END FUNCTION
```

*** STR$**

Questa funzione ritorna una stringa di lunghezza variabile il cui contenuto è espresso con caratteri ASCII, il numero indicato come argomento. È la funzione inversa della VAL e la sua sintassi è la seguente

STR$(espnum)

in cui espnum è il numero da convertire in stringa. La funzione ritorna sempre un carattere iniziale nella stringa, prima del numero convertito, che è un segno - se il numero era negativo, o uno spazio se era positivo.
Molte volte, se si ha a che fare solo con valori positivi, è necessario eliminare lo spazio posto all'inizio della stringa dalla funzione STR$; questo fatto è mostrato nel seguente esempio che fa uso di una funzione FNSTR$ definita dall'utente

```
DEF FNSTR$(X)=MID$(STR$(X), 2)
CLS
X=8
PRINT "LOG ("; STR$(X); ") = "; LOG(X)
PRINT "LOG ("; FNSTR$(X); ") = "; LOG(X)
END
```

*** STRIG**

Insieme alla funzione STICK, questa funzione è usata per controllare lo stato dei joysticks. La sintassi della funzione STRIG è

STRIG(funz)

in cui funz è un valore numerico compreso tra 0 e 7 al quale corrisponde un'azione determinata dalla seguente tabella

0 ritorna -1 se il tasto **inferiore** del joystick **A** è stato premuto dall'ultima esecuzione di STRIG(0); ritorna 0 altrimenti;

1 ritorna -1 se il tasto **inferiore** del joystick **A** è attualmente premuto, altrimenti 0;

2 ritorna -1 se il tasto **inferiore** del joystick **B** è stato premuto dall'ultima esecuzione di STRIG(2); ritorna 0 altrimenti;

3 ritorna -1 se il tasto **inferiore** del joystick **B** è attualmente premuto, altrimenti 0;

4 ritorna -1 se il tasto **superiore** del joystick **A** è stato premuto dall'ultima esecuzione di STRIG(4); ritorna 0 altrimenti;

5 ritorna -1 se il tasto **superiore** del joystick **A** è attualmente premuto, altrimenti 0;

6 ritorna -1 se il tasto **superiore** del joystick **B** è stato premuto dall'ultima esecuzione di STRIG(6); ritorna 0 altrimenti;

7 ritorna -1 se il tasto **superiore** del joystick **B** è attualmente premuto, altrimenti 0.

Nelle precedenti versioni di Basic il test dei joystick doveva essere prima abilitato dall'istruzione STRIG ON e poteva essere disabilitato dalla STRIG OFF; a partire da QB 4.0. queste istruzioni non sono necessarie ma la loro presenza non costituisce un errore; esse, per garantire la compatibilità, sono semplicemente **eliminate** dal Quick Basic quando riconosciute.
Se la funzione STRIG è usata in programmi compilati sotto MS-DOS con il compilatore BC, è necessario usare lo switch /v o /w.

Il seguente programma attende finché non è stato premuto il tasto inferiore del joystick A (questo programma è valido solo se si dispone effettivamente del joystick)

```
CLS
DO WHILE NOT STRIG(0)
LOOP
DO WHILE STRIG(1)
LOOP
END
```

*** STRING$**

Questa funzione, che ritorna una stringa di lunghezza variabile, permette l'uso di due sintassi e cioè

STRING$(lungh, codASCII)

STRING$(lungh, esprstr)

Nella prima sintassi, il valore lungh definisce il numero di caratteri che ritornerà la funzione mentre il valore codASCII è il valore, secondo la tabella ASCII, del carattere che deve essere ritornato. Quindi se, ad esempio, si volesse una stringa fatta da 100 punti, si potrebbe usare la linea

```
PUNTI$ = STRING$(100, 46)
```

in cui 100 è la lunghezza richiesta dalla stringa e 46 il codice ASCII del punto. In alternativa, si può usare la seconda sintassi in cui varia soltanto il secondo argomento che può essere una qualsiasi espressione stringa; in questo modo, la stringa ritornata dalla funzione è formata da un numero 'lung' di caratteri tutti uguali al **primo** carattere di 'esprstr'. Allo stesso modo del precedente esempio, la linea di programma da usare con la seconda sintassi sarebbe

```
PUNTI$ = STRING$(100, ".")
```

Il valore di lungh può essere compreso tra 0 (viene restituita una stringa nulla) e 32767, mentre quello di codASCII è, naturalmente compreso tra 0 e 255. Il parametro esprstr invece, non può essere nullo perché, in questo caso, viene generato un errore.

Nel seguente esempio si può vedere l'uso della funzione STRING$ e del carattere ASCII numero 178, nella visualizzazione di codici a barre

```
DIM V(1 TO 5)
CLS
PRINT
PRINT "Immettere 5 valori compresi tra 0 e 100"
PRINT
FOR X=1 TO 5
    DO
            LOCATE X+3, 10
            PRINT X; ") ";
            INPUT "", V(X)
    LOOP WHILE V(X)<0 OR V(X)>100
NEXT X
MAX=V(1)
FOR X=2 TO 5
    IF V(X)>MAX THEN
            MAX=V(X)
    END IF
NEXT X
CLS
FOR X=1 TO 5
    LOCATE X+10, 5
    BL=70*V(X)/MAX
    PRINT STRING$(BL, CHR$(178));
    PRINT USING " (###)"; V(X)
NEXT X
END
```

*** TAB**

Questa funzione, come la SPC, può essere usata solamente con le istruzioni PRINT e LPRINT e, nella sua sintassi, è obbligatorio l'uso del simbolo ; alla sua destra

TAB(n);

Essa è usata per spostare il cursore alla colonna 'n' di stampa (su video o su carta), per stampare i dati seguenti. Ad esempio, per visualizzare sulla riga corrente del video, alla colonna 44, la frase "Quick Basic", basta la seguente linea di programma

```
PRINT TAB(44); "Quick Basic"
```

Il valore massimo di 'n' dipende dal numero massimo di colonne a disposizione del dispositivo di output (normalmente 80 per il video e per la stampante; possono essere cambiate in 40 per il video e 136 per la stampante; altri valori possono essere disponibili per dispositivi di output non standard). Se il valore specificato per 'n' è minore di 1, esso viene considerato automaticamente, uguale a 1. Se invece, il valore di 'n' è maggiore del numero massimo di colonne disponibili, esso assume il valore del risultato dell'espressione

n MOD maxlargh

e l'output viene spostato sulla prossima riga. Se, durante l'uso della funzione TAB, viene specificata con 'n', una colonna già superata dall'istruzione PRINT corrente, l'output viene spostato alla colonna indicata ma nella **prossima riga**. Il seguente esempio mostra come l'uso appropriato della funzione TAB permetta semplici effetti di movimento sul video

```
CLS
IN=1
FI=70
ST=1
DO
        FOR T=IN TO FI STEP ST
                PRINT TAB(T); "Quick Basic";
        NEXT T
        IN=IN XOR 71
        FI=FI XOR 71
        ST=-ST
LOOP WHILE UCASE$(INKEY$)<>"F"
END
```

Notare l'uso dell'operatore XOR per cambiare il valore 70 in 1 e viceversa; il programma viene bloccato con l'uso del tasto F.

*** TAN**

Funzione trigonometrica del QB usata per calcolare il valore della tangente di un angolo espresso in radianti. La sintassi della funzione è la seguente

TAN(angolo)

e la precisione del valore restituito è uguale a quella dell'argomento usato; per avere quindi un risultato in doppia precisione è sufficiente adottare un argomento in doppia precisione.
A differenza di altri interpreti e compilatori che non interrompevano il programma in esecuzione se occorreva un overflow della funzione TAN, Quick Basic emette un segnale di errore arrestando il programma.

Il valore ritornato dalla funzione TAN è uguale, naturalmente, a quello ritornato dalla divisione del seno per il coseno dell'angolo considerato, come dimostra il seguente esempio

```
CLS
FOR X=0 TO 3.1415 STEP .15
        T=TAN(X)
        D=SIN(X)/COS(X)
        PRINT USING "TAN(#.##) = +##.#######"; X; T;
        PRINT TAB(30); USING "SIN(#.##)/COS(#.##)= +##.#######"; X; X; D
NEXT X
END
```

*** TIME$**

Questa funzione ritorna una stringa contenente l'orario corrente prelevato dall'orologio interno del DOS, nel formato hh:mm:ss in cui le ore (hh) vanno da 0 a 23. Questo orologio può essere aggiornato da QB con l'istruzione TIME$ (da non confondersi con la funzione in esame).
La sintassi della funzione TIME$ non prevede argomenti come dimostra il seguente esempio d'uso

```
CLS
PRINT "Orario corrente "; TIME$
END
```

*** TIMER**

Questa funzione restituisce, continuamente aggiornato, il valore dei secondi passati dalla mezzanotte, con gli eventuali decimali.
È usata con l'istruzione RANDOMIZE per la variazione delle sequenze dei numeri casuali e, soprattutto, per misurare la velocità di esecuzione dei programmi, come dimostra il seguente esempio

```
' Provare in questa riga le seguenti
'     istruzioni una alla volta
'          DEFINT  A-Z
'          DEFSNG  A-Z
'          DEFDBL  A-Z
CLS
PRINT "Attendere …"
STA!=TIMER
FOR T=1 TO 10000
NEXT
```

```
STO!=TIMER
PRINT "Il programma viene eseguito in "; STO!-STA!; "secondi"
END
```

Infatti, usando nella prima riga diverse istruzioni di definizione dei tipi delle variabili, si sono ottenuti i seguenti tempi di esecuzione, che sono solo indicativi presi in assoluto perché relativi alla velocità dell'hardware, ma che dimostrano le differenti velocità di trattamento dei dati numerici (i dati interi sono **molto** più veloci degli altri)

DEFINT A-Z	0.109375 secondi
DEFSNG A-Z	4.230469 secondi
DEFDBL A-Z	4.558594 secondi

*** UBOUND**

Vedere LBOUND

*** UCASE$**

Vedere LCASE$

*** USING**

Vedere PRINT USING, LPRINT USING, PALETTE USING

*** VAL**

Questa funzione ritorna il valore numerico corrispondente alla stringa usata come argomento. La sua sintassi è

VAL(esprstr)

Se il numero, all'interno della stringa, è preceduto da altri caratteri non numerici, la funzione ritorna il valore zero.
Il valore convertito può presentare qualche errore, per problemi di precisione nella conversione tra decimale e binario, se direttamente presentata a video con una istruzione PRINT; è possibile, per lo più, aggiustare tali errori, assegnando tale valore ad una variabile prima di stamparne il contenuto.
Ecco infatti, nel seguente programma di esempio, come si presenta tale caso

```
CLS
PRINT VAL("8.889")
A=VAL("8.889")
PRINT A
END
```

*** VARPTR**

La funzione VARPTR, insieme alla VARSEG, fornisce l'indirizzo in memoria di una variabile di qualsiasi tipo. L'indirizzo stesso è formato da due parti: il **segmento** ritornato da VARSEG e l'**offset** restituito da VARPTR. Per accedere direttamente con l'istruzione POKE e la funzione PEEK al contenuto delle variabili, è sufficiente preparare il segmento con l'istruzione DEF SEG ed il risultato di VARSEG, e poi adottare nelle suddette istruzioni, l'indirizzo fornito da VARPTR.

Mentre per quanto riguarda le variabili numeriche è possibile procedere come specificato, per le variabili alfanumeriche ciò non è vero dato che VARPTR e VARSEG, in questo caso, ritornano l'indirizzo del descrittore della stringa e non del suo contenuto (per maggiori informazioni vedere la funzione SADD).

Se si trattano degli arrays dinamici, **è necessario** l'uso della funzione VARSEG per determinarne il segmento. Nell'uso di queste funzioni (e anche di SADD) potrebbero esserci dei problemi dovuti al fatto che il Quick Basic riorganizza continuamente, per vari motivi, i dati in memoria e quindi questi potrebbero, ad un certo momento, non trovarsi all'indirizzo che si era ricavato in precedenza; per evitare tali problemi è **fortemente consigliabile** utilizzare gli indirizzi forniti dalle funzioni subito, senza effettuare prima altre operazioni.

Il programma di esempio dimostra come viene conservato in memoria un array di interi (l'indirizzo di un array è quello del suo primo elemento)

```
DEFINT A-Z
DIM EL(1 TO 15)
CLS
FOR X=1 TO 15
  EL(X)=X*100
NEXT X
DEF SEG=VARSEG(EL(1))
FOR X=VARPTR(EL(1)) TO VARPTR(EL(1))+ (15-1)*2 STEP 2
  PRINT USING "Elem. ## -> ### (L)  ### (H)"; X, PEEK(X); PEEK(X+1)
NEXT X
DEF SEG
END
```

la parte alta (H) vale 256 volte quella bassa (L), quindi il terzo valore, che ha parte bassa uguale a 44 e parte alta uguale a 1, vale 256*1+44 e cioè 300.

*** VARPTR$**

La funzione VARPTR$ ritorna una stringa contenente, in forma codificata, l'indirizzo di una variabile stringa ad uso delle istruzioni PLAY e DRAW quando usate con il sottocomando X (esecuzione di una sottostringa).

Mentre nelle precedenti versioni di BASIC, l'istruzione PLAY che doveva eseguire una sottostringa usava la seguente istruzione

```
F$="CD"
PLAY "XF$"
```

con QB, il programma deve essere modificato nel seguente modo

```
F$="CD"
PLAY "X" + VARPTR$(F$)
```

La sintassi della funzione

VARPTR$(nomevar)

indica chiaramente che il nome della variabile di cui si vuole l'indirizzo codificato in stringa, è l'argomento della funzione.

In questo esempio, si dimostra l'uso della funzione VARPTR$, facendo eseguire dall'istruzione PLAY la sottostringa che suona l'inizio di un celebre motivetto (il ciclo finale è usato per pemettere di terminare correttamente l'esecuzione dell'ultima nota, se il programma viene eseguito sotto MS-DOS)

```
F$="L8CCDCFECCDCGF"
PLAY "X" + VARPTR$(F$)
FOR X=1 TO 2000
NEXT X
END
```

*** VARSEG**

Vedere VARPTR

CAPITOLO 6

Le basi per programmare in Quick Basic

CAPITOLO 6

6.1 LE BASI PER PROGRAMMARE IN QUICK BASIC

6.1.1 Tipi di dati in Quick Basic

Il Quick Basic è in grado di trattare dati, sia di tipo numerico che alfanumerico (stringhe) e lo fa usando dei tipi di dati ben definiti (tipo di dati **elementari**). Essi sono, per le stringhe

- stringhe di lunghezza variabile
- stringhe di lunghezza fissa

e per numeri

- interi
- interi lunghi
- valori in virgola mobile in singola precisione
- valori in virgola mobile in doppia precisione

Per quanto riguarda le stringhe, esse possono essere di **lunghezza variabile** quando non è stata dichiarata in precedenza l'area occupata dalle stesse. In questo caso, la loro lunghezza può variare da zero (stringhe nulle) a 32767 caratteri il cui codice può variare da 0 a 255 secondo il codice ASCII esteso. Le variabili stringa sono identificate dal simbolo **$** tranne che non sia stata usata l'istruzione DEFSTR, mentre le costanti alfanumeriche sono caratterizzate dalle virgolette (*) iniziali e finali che le delimitano. Le variabili stringa a lunghezza variabile, inoltre, possono essere dichiarate con una frase di questo tipo

DIM S AS STRING

per mezzo della quale si rende inutile l'uso del simbolo $ per identificare la variabile stringa S.

Se la stessa frase si scrivesse

DIM S AS STRING * n

in cui 'n' è un numero intero compreso tra 0 e 32767, si definirebbe così una stringa **a lunghezza fissa** il cui spazio è allocato dal compilatore durante la compilazione e non durante l'esecuzione del programma che la usa.
La funzione LEN ritorna, per le stringhe a lunghezza fissa, sempre il valore usato nella loro dichiarazione, a prescindere dall'effettivo numero di caratteri contenuti nella stringa stessa. Per ricavare tale valore, in questo caso, è necessario usare la funzione RTRIM$ che permette di non considerare tutti gli spazi a destra della stringa aggiunti dal compilatore; il seguente programma di esempio, infatti, visualizza il numero 80 (lunghezza della stringa indicata nella dichiarazione) ed il numero 11 (lunghezza effettiva della stringa)

```
DIM S AS STRING * 80
S="Quick Basic"
CLS
PRINT LEN(S)
PRINT LEN(RTRIM$(S))
END
```

Con le stringhe a lunghezza variabile, invece, non c'è bisogno della funzione RTRIM$ dato che la funzione LEN ritorna sempre il numero effettivo di caratteri contenuti nella stringa. Bisogna comunque precisare che, per quanto riguarda l'occupazione di memoria, i due tipi di stringhe si comportano in maniera differente; infatti, mentre le stringhe a lunghezza la variabile occupano un numero di bytes equivalente al numero di caratteri effettivi contenuti nella stessa più 4, quelle a lunghezza fissa occupano il numero di caratteri contenuti nella stessa.
Se si tenta di assegnare una costante alfanumerica ad una variabile stringa a lunghezza fissa tale che questa costante sia più lunga della lunghezza dichiarata per la stringa, i caratteri in più saranno, automaticamente, ignorati e non faranno parte della stringa stessa. Il seguente esempio mostra tale evenienza

```
DIM S AS STRING * 5
CLS
S="Quick Basic"
PRINT S
END
```

infatti, verrà stampata la parola **Quick** dato che solamente i primi 5 caratteri della costante saranno assegnati alla variabile.
L'uso delle stringhe a lunghezza fissa è conveniente per la definizione di nuovi tipi di dati non elementari (v. 6.1.2), utili in molte occasioni, e necessari per una migliore gestione dei file random.

Per quanto riguarda, invece, i valori numerici **interi**, essi sono conservati in due bytes in cui, i primi 15 bits formano il numero ed il sedicesimo bit rappresenta il segno (secondo la notazione in complemento a due). In base a quanto detto, l'intervallo di valori così rappresentabile è compreso tra -32768 e +32767. Le variabili e le costanti intere sono caratterizzate dal simbolo **%**, non necessario se si usa l'istruzione DEFINT.
Gli **interi lunghi** invece, utilizzano 32 bits, di cui 31 per il numero ed uno per il segno, con lo stesso metodo dei precedenti, e così è possibile rappresentare tutti i valori interi compresi tra -2147483648 e +2147483647. Il nome delle variabili e delle costanti intere lunghe, è seguito dal segno **&** (cosa non possibile in BASICA e GW-BASIC, dato che, per questi, non esistono gli interi lunghi), cosa che può essere evitato usando l'istruzione DEFLNG.

Quest'ultimo tipo di dato numerico è stato introdotto con il Quick Basic e quindi non esiste un corrispondente nel BASICA o nel GW-BASIC; con questi interpreti BASIC è necessario usare il tipo in virgola mobile a singola precisione quando si devono rappresentare valori, anche interi, che non rientrano nell'intervallo -32768..+32767; anche se ciò non comporta un maggior impegno di memoria (sia gli interi lunghi che i valori in singola precisione occupano 4 bytes),

l'uso degli interi lunghi, quando possibile, rende l'elaborazione molto più veloce dato che i calcoli sono sviluppati tramite routines interne molto più semplici ed efficienti.
È buona norma quindi, scegliere sempre il tipo di dato **sufficiente** a contenere il valore da elaborare sia per risparmiare spazio ma, specialmente, per sveltire l'elaborazione stessa. È d'uso inserire, all'inizio di un programma (o di una sottoroutine), la seguente riga

DEFINT A-Z

che definisce tutte le variabili di tipo intero, per default, e non in singola precisione. È infatti dimostrato che, tranne per alcune eccezioni, nei programmi si fa molto più uso di valori interi che rientrano nell'intervallo -32768..+32767, che di altri tipi di dati numerici (si pensi ai cicli FOR..NEXT e alle variabili usate come contatori). Il seguente programma

```
CLS
T!=TIMER
FOR X=1 TO 30000
NEXT X
PRINT TIMER-T!
END
```

eseguito (con un AT 12 MHz e con il QB 4.0. dato che il QB 4.5. è un po' più veloce con i valori in virgola mobile), **senza** la suddetta riga, in 14.6 secondi, mentre, semplicemente aggiungendo la DEFINT A-Z, in soli 0.328 secondi e cioè più di **40** volte prima! Ecco perché, nella libreria BPLUS ed in molti esempi, è stata usata tale istruzione ed è stato indicato nel nome, con l'appropriato simbolo, il tipo di tutte le variabili non intere.

I valori in virgola mobile in singola precisione, sia costanti che conservati in una variabile, sono caratterizzati dal simbolo **!** ed occupano 4 bytes, come gli interi lunghi, ma con un formato diverso sia da questi che dagli equivalenti in BASICA e GW-BASIC. Infatti, questi ultimi linguaggi usano il formato Microsoft Binary a differenza del QB che usa il formato IEEE; tutto ciò si traduce in una incompatibilità, a livello di dati numerici in virgola mobile, conservati nei file random con le funzioni MKS$ e MKD$ dai linguaggi suddetti. Per garantire quindi, la portabilità di tali dati, sono state introdotte in Quick Basic le funzioni CVSMBF, CVDMBF, MKSMBF$ e MKDMBF$ compatibili con quelle degli altri Basic. Si noti che esistono le funzioni per la singola e la doppia precisione dato che tutto ciò vale anche per questo tipo di dato numerico. I numeri in singola precisione possono rappresentare, con una precisione fino alla settima cifra decimale, valori compresi in questi intervalli, sapendo che, i valori che si approssimano allo zero più di quelli indicati, sono considerati eguali a zero e, quelli che non rientrano nei limiti esterni, non possono essere rappresentati con tale tipo di dati

-3.402823E+38 -1.40129E-45 0 +1.40129E+45 +3.402823E+38

I valori in virgola mobile in doppia precisione, sono caratterizzati dal simbolo **#** ed occupano 8 bytes di memoria ciascuno; questo perché l'intervallo di valori rappresentabili è più ampio e la precisione si spinge fino alle 15 cifre decimali.
Anche per questi esiste il problema del formato (IEEE e non Microsoft Binary) discusso per i precedenti e gli intervalli in cui possono essere utilizzati, sono i seguenti

6.1.2 Tipi di dati definiti dall'utente

Oltre ai dati elementari, è possibile definire in Quick Basic, a differenza di altri tipi di Basic e come linguaggi come il C, Pascal e Macroassembler, dei **tipi di dati strutturati** composti dai tipi elementari suddetti. Per questo è presente in QB la coppia di istruzioni TYPE ed END TYPE tra le quali devono essere definite tutte le variabili (con il loro tipo) che faranno parte della struttura. Ad esempio, per definire un tipo di dato che possa contenere una **data**, si dovrà scrivere il seguente testo

```
TYPE UnaData
        Giorno AS INTEGER
        Mese AS INTEGER
        Anno AS INTEGER
END TYPE
```

Viene così creato un nuovo tipo di dato, chiamato **UnaData**, composto da 3 elementi interi, chiamati Giorno, Mese ed Anno, che potrà essere usato nelle dichiarazioni delle variabili che saranno fatte **dopo** tale dichiarazione. È quindi possibile creare una variabile con questa linea di programma

```
DIM DataNascita AS UnaData
```

in modo che la variabile DataNascita venga intesa dal QB, di **tipo composto** e cioè formata da più elementi semplici. Per usare tale variabile, sarà sufficiente specificare il suo nome e il nome dell'elemento da trattare separati da un punto; ad esempio, in questa maniera sarà assegnato il valore 1964 all'elemento Anno della variabile DataNascita

```
DataNascita.Anno = 1964
```

ed in quest'altro modo, sarà possibile visualizzare i tre componenti della variabile

```
PRINT DataNascita.Giorno
PRINT DataNascita.Mese
PRINT DataNascita.Anno
```

Anche delle variabili di tipo composto, come la nostra DataNascita, è possibile calcolare la lunghezza, in questo modo

```
PRINT LEN(DataNascita)
```

che risulterà uguale alla somma delle lunghezze di tutti gli elementi elementari che la compongono. Ma non sarà possibile trattare **tutti** gli elementi **contemporaneamente** riferendosi solo alla suddetta variabile, come nella seguente riga

```
PRINT DataNascita
```

dato che, così, il QB emetterà un apposito messaggio d'errore; sarà possibile solamente **assegnare** ad una variabile di tipo utente il contenuto di un'altra variabile utente dello stesso tipo e cioè in questo modo

```
DIM DataNascita AS UnaData
DIM Oggi AS UnaData
Oggi = DataNascita
```

Una volta dichiarato un nuovo tipo di dato, è possibile usarlo **anche** all'interno di una dichiarazione di un altro tipo di dato utente, come tipo di uno dei suoi elementi. Ad esempio, se si volesse usare il precedente tipo di dato (UnaData) per definirne un altro (UnaPersona), si potrebbe procedere nel seguente modo

```
TYPE UnaData
        Giorno AS INTEGER
        Mese AS INTEGER
        Anno AS INTEGER
END TYPE

TYPE UnaPersona
        Cognome AS STRING * 20
        Nome AS STRING * 20
        Sesso AS STRING * 1
        DataNascita AS UnaData
END TYPE
```

così che sarebbe ora possibile definire una variabile di tipo UnaPersona, con la seguente riga

```
DIM Persona AS UnaPersona
```

In questo caso, la lunghezza della variabile in questione è sempre data dalla somma delle lunghezze dei suoi elementi, considerando che la lunghezza di un elemento di tipo utente è, a sua volta data dalla somma della lunghezza degli elementi primitivi che lo compongono. in sostanza, per calcolare la lunghezza (e quindi l'occupazione in bytes) della variabile Persona, si devono considerare i seguenti valori

20 bytes per l'elemento Cognome
20 bytes per l'elemento Nome
1 byte per l'elemento Sesso
6 bytes per l'elemento DataNascita
        (dati dalla somma di 2 bytes per l'elemento Giorno)
        (2 bytes per l'elemento Mese)
        (2 bytes per l'elemento Anno)

per un totale di 20+20+1+6, **47** bytes, che è esattamente il valore ritornato dalla linea

```
PRINT LEN(Persona)
```

Nell'uso di tali dati notare che

a) l'uso degli elementi di tipo stringa a lunghezza variabile, non è possibile;

b) non è possibile inoltre usare degli arrays all'interno della dichiarazione di tipo utente, ma solo delle variabili semplici;

c) le dichiarazione dei dati utente che sono utilizzati in altre dichiarazioni, devono essere fatte **prima** di quest'ultime, per ovvii motivi;

d) non è sufficiente dichiarare i tipi utente per poterli utilizzare ma una o più variabile di quel tipo; non sono quindi le strutture TYPE..END TYPE che preparano lo spazio in memoria per i dati, ma le frasi DIM seguenti.

Il seguente programma di esempio, mostra come è possibile definire, assegnare e visualizzare dati con una variabile di tipo utente, rifacendosi agli esempi mostrati in precedenza

```
TYPE UnaData
        Giorno AS INTEGER
        Mese AS INTEGER
        Anno AS INTEGER
END TYPE

TYPE UnaPersona
        Cognome AS STRING * 20
        Nome AS STRING * 20
        Sesso AS STRING * 1
        DataNascita AS UnaData
END TYPE

TYPE UnImpiegato
        Persona AS UnaPersona
DataAssunzione AS UnaData
Livello AS INTEGER
        Stipendio AS LONG
END TYPE

DIM Impiegato AS UnImpiegato
CLS

Impiegato.Persona.Cognome = "Rossi"
Impiegato.Persona.Nome = "Paolo"
Impiegato.Persona.Sesso = "M"
Impiegato.Persona.DataNascita.Giorno = 1
Impiegato.Persona.DataNascita.Mese = 2
```

```
Impiegato.Persona.DataNascita.Anno = 1960
Impiegato.Persona.DataAssunzione.Giorno = 5
Impiegato.Persona.DataAssunzione.Mese = 11
Impiegato.Persona.DataAssunzione.Anno = 1979
Impiegato.Livello = 5
Impiegato.Stipendio = 1560000

PRINT Impiegato.Persona.Cognome
PRINT Impiegato.Persona.Nome
PRINT Impiegato.Persona.Sesso = “M”
PRINT Impiegato.Persona.DataNascita.Giorno
PRINT Impiegato.Persona.DataNascita.Mese
PRINT Impiegato.Persona.DataNascita.Anno
PRINT Impiegato.Persona.DataAssunzione.Giorno
PRINT Impiegato.Persona.DataAssunzione.Mese
PRINT Impiegato.Persona.DataAssunzione.Anno
PRINT Impiegato.Livello
PRINT Impiegato.Stipendio

PRINT
PRINT “Lunghezza record : “; LEN(Impiegato)
END
```

Notare che, per riferirsi ad un elemento di un tipo utente definito in un altro tipo utente, è sufficiente separare il nome della variabile e degli elementi stessi con un punto, come nella seguente riga

```
Impiegato.Persona.DataNascita.Anno
```

Per l’uso dei dati di tipo utente insieme ai file random (usato in sostituzione delle istruzioni FIELD, LSET e RSET), e per altre informazioni riguardo la TYPE..END TYPE, si rimanda alla descrizione dell’istruzione GET (per I/O da file) e della TYPE (v. 5.1.1).

6.1.3 Costanti e variabili

I dati, sia numerici che alfanumerici, possono essere rappresentati come costanti e contenuti in variabili. Le costanti alfanumeriche sono racchiuse tra virgolette, come nei seguenti esempi

“Paolo Rossi”
“Via Roma, 22”
“Quick Basic Compiler”

e le virgolette **non fanno parte** della costante.

Le costanti numeriche possono apparire nei seguenti formati

Tipo costanti	**Esempi**

• Interi in notazione decimale	55, +90, -100
• Interi in notazione esadecimale	&HAB, &H37C, &HFFF
• Interi in notazione ottale	&O661, &O23, &O100
• Interi lunghi in notazione decimale	67500, -50000
• Interi lunghi in notazione esadecimale	&HFFFFFFF&, &H90&
• Interi lunghi in notazione ottale	&O54324545&, &1&
• Valori decimali in virgola fissa in singola precisione	99.1287, -12.872
• Valori decimali in virgola fissa in doppia precisione	1.90#, 1.767667676
• Valori decimali in virgola mobile in singola precisione	12.2E9, 1.88E19
• Valori decimali in virgola mobile in doppia precisione	99.12D-8, -22D200

Come si può vedere dai precedenti esempi, le costanti possono essere positive e negative, espresse in decimale (senza alcun simbolo aggiuntivo), in esadecimale (facendo precedere i caratteri &H) e in ottale (facendo precedere il simbolo & o i caratteri &O) e può essere loro assegnato un **tipo elementare**. Infatti, mentre la costante esadecimale &HFF (che vale 255), la costante &HFFF& è intesa intera lunga; perché si comprenda come la cosa potrebbe fare una differenza in un programma, si esegua la seguente riga

```
PRINT &HFF * &HFF
```

Essa darà un errore di Overflow (superamento delle capacità aritmetiche del tipo di dato prescelto) perché, anche se i due valori possono essere rappresentati con un valore intero (255 è infatti nell'intervallo -32768..+32767), il risultato dell'operazione no (65025 è oltre tale intervallo). Dato che il QB converte il risultato nel tipo di dato più 'capace' che compare nell'espressione, questa deve essere modificata in questa maniera per potere funzionare

```
PRINT &HFF& * &HFF
```

Infatti, essendo uno dei termini **dichiarato espressamente** di tipo intero lungo (simbolo & finale), il risultato sarà calcolato come intero lungo.
Anche in quest'altro caso

```
PRINT 20 / 3
```

in cui il risultato (6.666667) è calcolato in singola precisione dato che è il minimo tipo di dato utile per la rappresentazione del risultato, le cose possono essere cambiate se si dichiara uno dei due termini, di tipo doppia precisione, in questo modo

```
PRINT 20 / 3#
```

In tal modo il risultato sarà in doppia precisione e, quindi, con 15 cifre decimali significative (6.666666666666667).

Le costanti possono essere anche dichiarate tramite l'istruzione CONST, appositamente inserita nel Quick Basic, per permettere l'uso di costanti figurative come già fatto da altri linguaggi. Il nome della costante viene formato come per il nome di una variabile (v. in seguito). È possibile, ad esempio, la definizione di una costante alfanumerica nel seguente modo

```
CONST LINEA = "_______________"
```

Da notare il fatto che il tipo di dato attribuito alla costante LINEA è insito nell'espressione e quindi non è necessario fare seguire alcun simbolo al nome della costante.

Il seguente programma mostra, invece, l'uso della costante numerica pigreco nel calcolo della circonferenza e dell'area di un cerchio

```
CONST PIGRECO=3.141592
CLS
INPUT "Raggio : ", R
PRINT
PRINT "Circonferenza = "; R*2*PIGRECO
PRINT "Area = "; R*R*PIGRECO
END
```

Notare i seguenti fatti

- non è possibile usare delle variabili con il nome di costanti figurative già esistenti e definite con una istruzione CONST;

- non basta definire una costante di tipo in doppia precisione aggiungendo il simbolo # al nome (PIGRECO#) per ottenere risultati corretti in doppia precisione; anzi, ciò non è necessario, dato che il tipo di costante viene assegnato dal QB automaticamente a seconda del tipo di espressione che segue; è alla costante 3.141592 che bisogna aggiungere tale simbolo (3.141592#) per definire PIGRECO in doppia precisione ed ottenere così risultati con tale tipo di dato.

Le **variabili** in Quick Basic sono dei 'contenitori' in cui potere inserire dei dati, sia numerici che alfanumerici. Esse hanno un nome formato da **1** a **40 caratteri**; il primo di questi deve essere una lettera, mentre gli altri possono essere lettere e cifre; nessun altro carattere può fare parte del nome, se si esclude il 'punto' quando si trattano le variabili di tipo utente per trattarne i singoli elementi (v. 6.1.2).
Esse possono esseredi un determinato tipo (v. 6.1.1); possono essere di tipo primitivo o utente, numerico o alfanumerico. Il tipo delle variabili può essere dichiarato tramite un apposito simbolo alla fine del loro nome o con una frase DIM, COMMON, REDIM, SHARED o STATIC (v. tutte queste istruzioni in 5.1.1). Nella conseguente tabella si notano i simboli e le frasi di dichiarazione adatti ad ogni tipo di variabile

Tipo	Simbolo	Dichiarazione DIM..AS
Intero	%	INTEGER

Intero Lungo	&	LONG
Sing. Precisione	!	SINGLE
Doppia Precisione	#	DOUBLE
Stringa lungh. var.	$	STRING
Stringa lungh. fissa	$	STRING * n

Il nome delle variabili è soggeto alle seguenti regole

- ai fini dell'identificazione delle variabili, non importa se queste sono scritte con lettere minuscole o maiuscole; le variabili 'IVA', 'Iva' e 'iva' sono del tutto equivalenti;

- deve essere diverso da tutte le parole chiave (le istruzioni e funzioni) del linguaggio, anche se queste possono fare parte del nome stesso; è cioè vietato l'uso di una variabile chiamata 'Data' ma è possibile usare la variabile 'DataNascita';

- il nome di una variabile non può iniziare per FN dato che verrebbe scambiata per una funzione definita con la DEF FN…;

- il nome delle variabili deve essere distinto dal nome delle Subs, delle Functions e delle costanti definite con l'istruzione CONST in **tutta la procedura**;

- le variabili di nome uguale, ma di diverso tipo, possono essere usate e sono distinte;

- se si dichiara una variabile con una AS …, è necessario riferirsi, in altre situazioni, a questa variabile con la stessa dichiarazione AS …

Sono nomi validi di variabili Iva19, Totale, TotaleGenerale, Nominativo$, Eta, DataNascita, Contatore2; sono invece errati questi altri nomi

Eta' (compare un simbolo illegale)

Data di nascita (non sono ammessi spazi nel nome)

1Xfer04 (inizia con una cifra)

Girogirotondoquantoebelloilmondogiralaterra (più di 40 caratteri)

Il tipo delle variabili, per default, è la singola precisione, ma questo stato di cose può variare se si usano le istruzioni DEFINT, DEFLNG, DEFSNG, DEFDBL, DEFSTR, anzi, l'uso della DEFINT è

consigliabile (v. 6.1.1). Se si usano tali istruzioni **non** è più necessario il simbolo per le variabili il cui tipo è stato definito di default; se, cioè, si dovesse eseguire l'istruzione

```
DEFSTR A-Z
```

definendo così tutte le variabili di tipo stringa, le seguenti righe

```
S = "Questa e' una stringa"
PRINT S
```

sarebbero del tutto corrette.

Alle variabili può essere assegnato un valore, nei seguenti modi

- con un valore costante (X=2.9, N$="Paolo")

- con un'altra variabile (X=Y, X$=N$)

- con una espressione (X=3*D, X$="Paolo"+N$)

è però necessario che il tipo del risultato dell'espressione posta alla **destra** del segno uguale sia concorde con quello della variabile posta alla **sinistra** dello stesso; eventuali conversioni di tipo possono essere fatte con le funzioni VAL e STR$.

6.1.4 Variabili globali, locali, statiche ed automatiche

Tutte le variabili, in un programma scritto in BASICA o in GW-BASIC, esistono dal momento in cui sono create fino a quando termina il programma stesso. Esse sono, inoltre, comuni a tutte le sottoroutines, senza distinzione; si può dire che esse sono tutte **globali**.
Anche in Quick Basic è possibile dichiarare delle variabili e delle costanti simboliche **globali**; basta, per le costanti, usare l'istruzione CONST a livello del modulo e, per le variabili, usare l'attributo **SHARED** nelle istruzioni DIM, REDIM e COMMON. Nel seguente programma

```
CONST PIGRECO = 3.141592
DIM SHARED Raggio AS SINGLE
```

sono dichiarate globali la costante PIGRECO (nella prima riga) e la variabile Raggio (nella seconda riga); da notare il fatto che, in questa maniera, il loro valore diventa disponibile a **qualsiasi** Sub o Function.
Le costanti e le variabili **locali** invece, sono valide solamente all'interno della Sub, Function o modulo in cui vengono dichiarate. Le costanti dichiarate con l'istruzione CONST all'interno di Sub o Function sono locali a questa. Tutte le variabili usate all'interno di una Sub o Function sono locali, se non diversamente specificato nel modulo. Ad esempio, il programma

```
' (Main Module)
CLS
DIM X AS INTEGER
X=3.14
```

```
PRINT "X nel Main : "; X
CALL Prova
PRINT "X nel Main : "; X
END
' (Sub Prova)
SUB Prova
    X=1.5
    PRINT "X nella Sub : "; X
END SUB
```

produrrà il seguente output

```
X nel Main :  3
X nella Sub :  1.5
X nel Main :  3
```

a dimostrare che la variabile X della Sub è **diversa** da quella definita nel Main. Questo modo di operare garantisce che le variabili usate in una Sub o Function non siano influenzate e non influenzino quelle usate nel Main; così le Function e le Sub diventano delle **entità autonome**, come delle 'scatole' nelle quali si immettono dei dati e se ne ottengono altri senza alcuna influenza; è questo il modo di operare di altri linguaggi di programmazione quali C e Pascal. Si possono costruire, quindi, gruppi di Functions e Subs, indipendenti, che possono usare tutte le variabili possibili, dato che sono locali.
Notare che, nell'esempio precedente, la variabile X dichiarata nel Main deve essere considerata locale a quest'ultimo e non globale; solamente l'attributo SHARED nella DIM nel Main può rendere la variabile X globale; è vietato l'uso dell'attributo SHARED nella frase DIM se questa è usata all'interno di una Function o di una Sub; al loro interno, infatti, non si possono dichiarare, ma solo usare se dichiarate nel Main, variabili globali.

L'istruzione **SHARED** può essere usata, invece, all'interno di una Sub o Function al fine di **condividere** una o più variabili. Queste non diventano globali, ma sono rese comuni a più Subs, Functions ed al Main. Il seguente esempio, modifica del precedente, mostra come ciò possa essere fatto

```
' (Main Module)
CLS
DIM X AS INTEGER
X=3.14
PRINT "X nel Main : "; X
CALL Prova1
PRINT "X nel Main : "; X
CALL Prova2
PRINT "X nel Main : "; X
END
' (Sub Prova1)
SUB Prova1
    SHARED X AS INTEGER
    PRINT "X nella Sub1 : "; X
```

```
END SUB
' (Sub Prova2)
SUB Prova2
    PRINT "X nella Sub2 : "; X
END SUB
```

Tale programma infatti, mostra che il valore della variabile X è uguale per il **Main** e per la Sub **Prova1**, ma non per la Sub Prova2 dato che l'istruzione SHARED, che mette in comune le variabili, non è presente in quest'ultima. Notare come questa istruzione permetta un funzionamento delle variabili che è una via di mezzo tra globale e locale. Essa permette di far condividere delle variabili definite nel Main, **solo** a certe Subs e Functions.

Le variabili locali possono essere, inoltre, **statiche** e **automatiche**. Esse sono, normalmente, automatiche, sono cioè create (ed azzerate) al momento del primo impiego o dichiarazione e sono 'distrutte' quando termina la Sub o Function che le usa. In altre parole, esse esistono (ed occupano memoria) solo finché serve, poi vengono eliminate (attenzione, non azzerate, ma eliminate; la memoria che occupavano, cioè, viene resa libera).
Ecco quindi che, chiamando due volte la stessa Sub o Function, non si ritrova il valore che la variabile locale aveva, in quanto questa viene creata nuovamente ad ogni chiamata della Sub.
Alcune volte è necessario che tale valore venga conservato tra una chiamata e l'altra e, per questo, la variabile deve essere dichiarata **statica**. A ciò provvede l'istruzione **STATIC** posta all'interno della Sub o Function, come mostra questo esempio

```
' (Main Module)
DECLARE SUB Prova( )
CLS
DIM X AS INTEGER
FOR X=1 TO 5
    PRINT "X in Main : "; X,
    CALL Prova
NEXT X
END
' (Sub Prova)
SUB Prova
    STATIC X AS INTEGER
    PRINT "X nella Sub : "; X, "Y nella Sub : "; Y
    X=X+3
    Y=Y+5
END SUB
```

Notare come la variabile X definita nel Main e quella nella Sub Prova, assumano valori diversi e distinti (in quanto X non è globale), ma che il valore della X della Sub sia conservato tra ogni chiamata della stessa, a differenza della Y. Per consentire ciò anche per la variabile Y, basta modificare la frase con l'istruzione STATIC, in questo modo

```
STATIC X AS INTEGER, Y AS INTEGER
```

Ma, dato che, in questo modo, **tutte** le variabili usate dalla Sub, sono dichiarate statiche, si può evitare la frase suddetta ed ottenere lo stesso scopo, riscrivendo la Sub così

```
' (Sub Prova)
SUB Prova STATIC
        PRINT "X nella Sub : "; X, "Y nella Sub : "; Y
        X=X+3
        Y=Y+5
END SUB
```

L'attributo STATIC subito dopo il nome della Sub, provvederà a rendere statiche tutte le variabili usate dalla stessa, senza altra indicazione.
Notare che l'uso di tale attributo, quando non sconsigliato, rende l'esecuzione delle Subs e delle Functions, un po' più veloce dato che così non si deve perdere tempo a creare e distruggere le variabili locali; è però senza dubbio che tale metodo impiega più memoria usata per conservare i valori delle variabili locali fino alla fine del programma.
Una eccezione al meccanismo delle variabili locali è costituita da quelle definite all'interno di una struttura DEF FN..END DEF; esse, infatti, sono sempre globali, tranne che non vengano definite statiche; una istruzione STATIC seguita da tutte le variabili che si devono rendere locali, basta a questo scopo.

Ricapitolando, si può dire che

- una variabile dichiarata a livello di modulo, con una DIM, REDIM o COMMON e con l'attributo SHARED, è **globale** e disponibile per tutte le Subs o Functions che ad essa si riferiscono;

- una costante simbolica dichiarata con l'istruzione CONST a livello di modulo, è **globale**; se dichiarata in una Sub o Function è **locale** a quest'ultima;

- una variabile che non è globale ed appare in una Sub o Function è **locale** a quest'ultima;

- l'istruzione SHARED permette di condividere delle variabili dichiarate a livello di modulo, con alcune Subs o Functions; essa **non** rende globali tali variabili;

- le variabili all'interno delle strutture DEF FN..END DEF, son tutte **globali** tranne quelle dichiarate con l'istruzione STATIC, che sono **locali**;

- tutte le variabili locali sono, normalmente, **automatiche**; esse esistono solo all'interno della Sub o Function in cui sono usate; al termine di quest'ultima vengono distrutte;

- le variabili definite con l'istruzione STATIC all'interno di una Sub o Function, non sono più automatiche ma **statiche**; esse conservano il loro valore tra diverse chiamate della Sub stessa; è possibile rendere

statiche **tutte** le variabili di una Sub o Function, aggiungendo l'attributo STATIC al nome di quest'ultima.

6.1.5 Gli arrays

Il Quick Basic prevede, come il BASICA ed il GW-BASIC, gli **arrays**; questi sono dei 'gruppi' di variabili, tutte dello stesso tipo, che sono identificate dallo stesso nome.
Ognuna di queste variabili viene chiamato **elemento** dell'array ed è identificata da uno, o più, numeri interi chiamati **indici** posti tra parentesi tonde; secondo il numero di indici di un array, si dirà che questo ha una, due o più **dimensioni**. In genere, un array ad una dimensione, viene indicato come **vettore**; è comunque preferibile, specificandone le dimensioni, continuare a chiamarlo array.

Per preparare in memoria lo spazio per gli arrays, esiste l'istruzione DIM (v. 5.1.1) tramite la quale si può specificare il nome, il tipo, le caratteristiche di visibilità (v. 6.1.4), il numero di dimensioni e di elementi per ogni dimensione, specificando inoltre, i valori minimi e massimi consentiti per gli indici. Ad esempio, il seguente array

```
DIM Tabella%(1 TO 20, 1 TO 50)
```

è stato chiamato Tabella, è intero, è locale, ha 2 dimensioni, ha 20 elementi nella prima e 50 nella seconda dimensione ed occupa, dopo un rapido calcolo, circa 2000 bytes. Esso ci permette di avere a disposizione 1000 variabili intere definite con due indici che possono variare da 1 a 20 e da 1 a 50; elementi di questo array sono i seguenti

```
Tabella%(5, 4)
Tabella%(20, 45)
Tabella%(12, 34)
```

e così via; ma non lo sono questi altri

```
Tabella%(0, 24)
Tabella%(28, 55)
Tabella%(20, 0)
```

Ogni elemento di un array, identificato dal nome e dagli indici, è praticamente, tranne alcune eccezioni, **equivalente** ad una variabile semplice. Esso può essere usato nelle espressioni

```
Iva = Imponibile(1, 6) * 19 / 100
```

con le struttura cicliche di controllo come la FOR (anche se non può essere posto come variabile di controllo)

```
FOR X=1 TO Massimi(6)
```

può essere usato con quasi tutte le istruzioni

```
PRINT Tabella%(10, 20)
```

Gli arrays possono essere di qualsiasi tipo (intero, intero lungo, singola precisione, doppia precisione, stringa a lunghezza fissa e variabile ed anche utente) e seguono le stesse regole esistenti per le variabili semplici, per l'attribuzione di quest'ultimo. È possibile, nella frase DIM indicare, dopo il nome, un simbolo, come per le variabili semplici, che indichi il tipo; oppure si può specificare, sempre nella DIM, con la clausola AS; i seguenti esempi sono equivalenti

```
DIM X$(1 TO 10)
DIM X(1 TO 10) AS STRING
```

Il numero massimo di indici (e quindi di dimensioni) per il QB è fissato in **60**, ma c'è da dubitare che un array di tal genere possa essere, facilmente, usato in un programma. Il numero massimo di elementi, invece, che è ogni dimensione può avere, è invece di **32767**, altro valore più che sufficiente per gli usi comuni.

Tuttavia, a differenza del BASICA e del GW-BASIC, il QB ammette **indici negativi** per gli arrays, da -32768 in poi, fino a 0 e a +32767. In altre parole, è possibile dimensionare un array del tipo

```
DIM Tabella(-10 TO 30)
```

Esso una una dimensione e 41 elementi ed occupa, circa, 82 bytes. Il numero di bytes occupati da un array non può comunque superare i 64K (anzi, qualcosa in meno, dato che esiste anche uno stack da considerare) e questo si può controllare considerando che, ogni elemento dell'array, a seconda del tipo, occupa 2, 4, 8 o un numero di bytes variabile. Questo limite è posto per gli arrays **statici** che, a differenza dei **dinamici**, occupano un solo segmento di memoria del sistema.
In mancanza della frase DIM, l'array di cui viene usato un elemento, è automaticamente dimensionato con 11 elementi, numerati da 0 a 10; questo fatto è comune a precedenti versioni di BASIC, ma è comunque, sconsigliabile ed è preferibile dimensionare **tutti** gli arrays usati, anche se piccoli, per maggior chiarezza.

Un'altra caratteristica che differenzia il QB da altri linguaggi BASIC, è la presenza della parola chiave TO all'interno della DIM. È infatti possibile specificare, come già fatto, un valore **minimo** oltre al **massimo** per un indice, cosa che era impossibile in precedenza. Il valore minimo **non** è comunque obbligatorio ed, in sua mancanza, viene assunto il valore zero. Le due righe seguenti, sono dunque equivalenti per il QB

```
DIM Tabella(0 TO 100)
```

e

```
DIM Tabella(100)
```

Tutto ciò è valido solo se non è stata usata l'istruzione **OPTION BASE 1**, altrimenti il valore minimo è inteso uguale ad uno; il seguente esempio mostra altre due righe equivalenti ma, questa volta, con la suddetta istruzione

```
DIM Tabella(1 TO 100)
```

e

```
OPTION BASE 1
DIM Tabella(100)
```

È possibile dichiarare degli arrays di tipo utente, per avere a disposizione un numero elevato di variabili di un certo tipo definito e ciò è possibile sempre tramite l'istruzione DIM, avendo cura di specificare, con la clausola AS, il tipo della variabile. Ad esempio, la frase DIM del programma in 6.1.2, può essere modificata nel seguente modo

```
DIM Impiegati(1 TO 200) AS Un Impiegato
```

per specificare un array ad una dimensione composto da 200 elementi di tipo utente. A questo punto, per il trattamento di tali elementi, basta specificare il valore dell'indice, subito dopo il nome dell'array, come nelle linee

```
Impiegato(1).Persona.Cognome = "Rossi"
Impiegato(1).Persona.Nome = "Paolo"
Impiegato(1).Persona.Sesso = "M"
Impiegato(1).Persona.DataNascita.Giorno = 1
Impiegato(1).Persona.DataNascita.Mese = 2
Impiegato(1).Persona.DataNascita.Anno = 1960
Impiegato(1).Persona.DataAssunzione.Giorno = 5
Impiegato(1).Persona.DataAssunzione.Mese = 11
Impiegato(1).Persona.DataAssunzione.Anno = 1979
Impiegato(1).Livello = 5
Impiegato(1).Stipendio = 1560000
```

Un uso molto frequente degli arrays si fa in matematica, quando si trattano le matrici. Ecco, ad esempio, un programma che esegue la somma di due matrici quadrate 10 x 10 e pone il risultato in un'altra matrice delle stesse dimensioni

```
DECLARE SUB MatAdd(OpA!(), OpB!(), Res!())
DECLARE SUB MatPrint(Mat!())
RANDOMIZE TIMER
DIM A(1 TO 10, 1 TO 10), B(1 TO 10, 1 TO 10), S(1 TO 10, 1 TO 10)
CLS
FOR R=1 TO 10
        FOR C=1 TO 10
                A(R, C)=INT(RND * 100)
                B(R, C)=INT(RND * 100)
        NEXT C
NEXT R
MatPrint A()
MatPrint B()
MatAdd A(), B(), S()
MatPrint S()
```

```
END
SUB MatAdd(OpA(), OpB(), Res())
        FOR R=LBOUND(OpA, 1) TO UBOUND(OpA, 1)
                FOR C= LBOUND(OpA, 2) TO UBOUND(OpA, 2)
                        Res(R, C)=OpA(R, C) + OpB(R, C)
                NEXT C
        NEXT R
END SUB
SUB MatPrint (Mat())
        PRINT
        FOR R=LBOUND(Mat, 1) TO UBOUND(Mat, 1)
                FOR C= LBOUND(Mat, 2) TO UBOUND(Mat, 2)
                        PRINT Mat(R, C);
                NEXT C
        NEXT R
END SUB
```

Da questo esempio si possono trarre diverse conseguenze, e cioè

- è possibile usare, per gli indici, delle variabili o delle espressioni il cui valore determina l'elemento a cui si vuole puntare; anche se le variabili sono di diverso tipo numerico, il risultato viene sempre convertito in intero (-32768..+32767), in quanto gli indici possono essere solo di questo tipo;

- si possono indicare degli interi array come argomenti di Subs e Functions; basta indicare, nella chiamata della Sub, come argomento, il nome dell'array seguito da una coppia di parentesi tonde senza indici; non è necessario indicare il numero delle dimensioni all'interno delle parentesi, come in altre versioni di BASIC;

- all'interno delle Subs e delle Functions, si possono rendere utili le funzioni LBOUND e UBOUND che forniscono, per un dato array ed una sua dimensione, il numero minimo e massimo di elementi presenti; esse sono utilizzabili con i nomi degli arrays usati come argomento per conoscere tali dati sugli arrays passati.

6.1.6 Il passaggio degli argomenti alle Subs e Functions

Gli argomenti che è possibile passare alle Subs ed alle Functions sono costituiti da costanti, variabili semplici, variabili utente, array ed espressioni. Essi devono apparire, nella lista di chiamata della Sub, nello stesso ordine di come sono nella lista di dichiarazione della Sub stessa ed i loro tipo deve essere coerente.
La seguente Sub, ad esempio

```
SUB EQUA2(ParA!m ParB!, ParC!)
```

può essere chiamata con la riga

```
CALL EQUA2(A!, B!, C!)
```

o con

```
EQUA2 A!, B!, C!
```

che è del tutto equivalente; in questo caso i tre valori delle variabili, sono passati ai tre argomenti della Sub e da questa elaborati. Anche i dati di tipo utente possono essere passati come argomenti, come dimostra questo programma

```
DECLARE SUB VisDNasc(DNasc AS ANY)
TYPE UnaData
    Giorno AS INTEGER
    Mese AS INTEGER
    Anno AS INTEGER
END TYPE
DIM DataNascita AS UnaData
CLS
DataNascita.Giorno = 18
DataNascita.Mese = 8
DataNascita.Anno = 1964
VisDNasc DataNascita
END
SUB VisDNasc (DNasc AS UnaData)
    PRINT USING "##/##/####"; DNasc.Giorno, DNasc.Mese, DNasc.Anno
END SUB
```

Gli argomenti possono essere passati in due maniere: per **riferimento** e per **valore**. Normalmente essi sono passati per riferimento: viene cioè passato l'**indirizzo** in memoria della variabile e non il suo valore in modo tale che la Sub, conoscendo la sua posizione, possa modificarne il valore. È questo il caso del prossimo esempio

```
DECLARE SUB Potenze(P1!, P2!, P3!, P4!)
CLS
INPUT "Valore : ", V
V2=0
V3=0
V4=0
Potenze V, V2, V3, V4
PRINT V, V2, V3, V4
END
SUB Potenze(P1!, P2!, P3!, P4!)
  P2=P1^2
  P3=P1^3
  P4=P1^4
END SUB
```

in cui le variabili V2, V3 e V4 sono passate per riferimento (cioè per indirizzo); così possono essere **modificate** dalla Sub.

Se si volesse passare una variabile per **valore** evitando così che la si possa modificare, il QB usa un metodo abbastanza semplice: copia la variabile in un'area temporanea e fornisce l'indirizzo di quest'ultima alla Sub; è così modificato il contenuto di tale area e non quello della variabile originaria. Tutto ciò è possibile tutte le volte che, come argomento viene passato il risultato di una **espressione**. Nel caso di una singola variabile, essa viene intesa come facente parte di una espressione se racchiusa tra parentesi. Modificando quindi, la riga della chiamata alla Sub, in questo modo

```
Potenze V, (V1), (V2), (V3)
```

il programma non funziona più, dato che la Sub, non conoscendone l'indirizzo, non può modificare le variabili suddette.

Alcune volte, questo è il metodo usato quando **non si vuole** che la variabile usata come argomento, venga modificata. Infatti, nel seguente esempio, il contenuto iniziale della variabile usata come contatore, non viene modificato solo se si adotta tale metodo

```
DECLARE SUB ScriviNVolte (N%, Frase$)
DEFINT A-Z
CLS
INPUT “Volte : “, Nv
LINE INPUT “Frase : “, Fr$
PRINT
ScriviNVolte (Nv), Fr$
PRINT
PRINT “La frase : “; FR$; “ e’ stata scritta “; Nv; “ volte”
END
SUB ScriviNVolte(N, Frase$)
        DO WHILE N>0
                PRINT Frase$
                N=N-1
        LOOP
END SUB
```

Il risultato dell'elaborazione è corretto perché, nella chiamata della Sub, la variabile Nv è stata racchiusa tra parentesi; il suo valore originario, così, non viene perso con i continui decrementi e l'ultima frase visualizzat, è corretta.

6.1.7 Le espressioni e gli operatori

Il QB mette a disposizione dell'utente un certo numero di operatori e delle precise regole per la formazione di espressioni di diverso tipo. Gli operatori sono raggruppabili in 4 insiemi distinti

- operatori aritmetici
- operatori relazionali

- operatori logici
- operatori su stringhe

Al primo gruppo appartengono il simbolo dell’addizione (**+**), quello della sottrazione (**-**), della moltiplicazione (*****), della divisione (**/**), della divisione intera (****), dell’elevamento a potenza, della negazione (**-**) e la parola chiave per il calcolo del modulo (**MOD**).
Al secondo gruppo, appartengono invece, il simbolo di maggiore (**>**), minore (**<**), uguale (**=**), minore-uguale (**<=**), maggiore-uguale (**>=**) e diverso (**<>**).
Gli operatori logici sono invece **NOT**, **AND**, **OR**, **XOR**, **EQV** ed **IMP**.
Dell’ultimo gruppo, infine, fanno parte il simbolo usato per l’unione di stringhe (**+**) e tutti gli operatori relazionali (<, >, =, <=, >=, <>) quando usati per confrontare stringhe.

Nell’uso degli operatori che operano sui valori numerici, esistono delle priorità ben definite; alcuni di essi, cioè, vengono eseguiti prima di altri, se non compaiono parentesi che ne alterano tale priorità. La seguente tabella mostra l’ordine di esecuzione di tali operatori, partendo da quelli con maggiore priorità

Elevamento a potenza	^
Negazione	-
Moltiplicazione e divisione	* /
Divisione intera	\
Modulo aritmetico	MOD
Addizione e sottrazione	+ -
Operatori relazionali	< > = <= >= <>
Operatori logici	NOT AND OR XOR EQV IMP

Solo se una negazione ed un elevamento a potenza sono vicini in una espressione, la negazione viene eseguita prima dell’elevamento e questa è l’unica eccezione alla regola della priorità; a parità di priorità, l’espressione viene calcolata da sinistra verso destra.

L’operatore di divisione intera (****), opera su valori che, in precedenza, sono arrotondati ad un intero. Il risultato fornito è anch’esso intero; ad esempio, le seguenti espressioni

```
PRINT 77 \ 4, 983 \ 23, 123.12 \ 2.3
```

producono i seguenti risultati

```
19      42      61
```

Allo stesso modo si comporta l’operatore **MOD**, ma esso ritorna il resto della divisione tra i suoi due operandi. La riga di programma seguente

```
PRINT 77 MOD 4, 983 MOD 23, 123.12 MOD 2.3
```

produce i seguenti risultati

```
1       17      1
```

che sono, appunto, i resti delle divisioni degli operandi nelle espressioni (dopo essere stati arrotondati ad interi).

Gli operatori relazionali sono usati per confrontare due valori numerici o due stringhe. Essi permettono di ottenere un risultato logico (vero o falso) a seconda della relazione proposta. Tale valore viene ritornato dal QB come intero (-1 = vero, 0 = falso). La seguente espressione, ritorna appunto, il valore -1, in quanto verificata

```
PRINT 77 > 10
```

mentre, quest’altra, 0, in quanto falsa

```
PRINT 19 + 22 < 10
```

Attenzione a scritture del tipo

```
A = B = C
```

che, se anche in certi linguaggi (come C) vogliono assegnare il valore della variabile C sia alla B che alla A, in QB non fanno altro che assegnare un valore vero (-1) o falso (0) alla variabile A, a seconda che B sia o meno uguale a C. Notare che è possibile sfruttare il valore ritornato dalle espressioni di relazione, per realizzare funzioni in una sola linea. Ad esempio, per ricavare il massimo tra due valori numerici, è sufficiente scrivere la seguente funzione

```
DEF FNMAX(V1, V2) = V1*-(V1>=V2) + V2*-(V1<V2)
```

Per capire come questa funzioni, bisogna esaminare le sue parti; delle due espressioni tra parentesi (V1>=V2) e (V1<V2), solamente una, per due determinati valori, è vera (e restituisce il valore -1), mentre l’altra è falsa (e restituisce il valore 0); ambedue sono negate dall’operatore posto davanti la parentesi; diventa quindi, 1 quella vera mentre quella falsa resta eguale a 0.

Questi due valori sono moltiplicati, ognuno a parte, per il primo numero (V1) e per il secondo (V2) così che, la parte dell’espressione vera, assume il valore del numero (1 * V1 oppure 1 * V2), mentre l’altra si annulla (0 * V1 oppure 0 * V2); la somma finale non fa altro che aggiungere 0 al risultato già ottenuto e quindi, non lo varia. Ad esempio, se V1 fosse 22 e V2 fosse 10, ecco i calcoli effettuati

(V1>=V2)	(V1<V2)
22>=10	22<10
vero	**falso**
-1	**0**
-(-1)	-(0)
1	**0**
V1*1	V2*0
22*1	10*0
22	**0**

22 + 0
22 (valore massimo trovato)

È bene fare molta pratica con le espressioni relazionali dato che esse hanno una grossa importanza per la realizzazione di corretti programmi in QB. Esse compaiono, infatti, con molte istruzioni molto importanti (IF..THEN..ELSE..END IF, DO..LOOP, WHILE..WEND) ed una loro approfondita conoscenza che non può che aiutare il programmatore.

Gli operatori logici sono usati per permettere dei tests tra relazioni multiple ed il loro uso è alquanto semplice.

L'operatore AND, ad esempio, è usato quando si vuole che il risultato di una espressione logica sia vera **solo** se sono vere tutte le parti di quest'ultima. Le righe seguenti, infatti

```
IF A>B AND A>C THEN
        PRINT "Il numero più grande e' "; A
END IF
```

dimostrano come è possibile 'legare' le due espressioni di relazione (A>B e A>C) in modo che ambedue debbano essere verificate per risultare vera anche l'espressione in generale.

Altri operatori logici possono essere usati ed il loro modo di funzionare è riassunto nella seguente tabella

A	B	A NOT A	A AND B	A OR B	A XOR B	A EQV B	A IMP B
F	**F**	**V**	**F**	**F**	**F**	**V**	**V**
F	**V**	**V**	**F**	**V**	**V**	**F**	**V**
V	**F**	**F**	**F**	**V**	**V**	**F**	**F**
V	**V**	**F**	**V**	**V**	**F**	**V**	**V**

Gli operatori logici possono agire anche a livello di bits su valori interi e interi lunghi. Ad esempio, le righe

```
DEFINT A-Z
CLS
H=10
W=4
PRINT H OR W
END
```

producono, come risultato, il valore 14; come questo viene ottenuto, è semplice da capire, se si considerano i valori in binario e si applica la tabella della verità dell'operatore OR, bit per bit

```
variabile H (10)        0000000000001010
variabile W (4)         0000000000000100
                        --------------------------
```

risultato (14)   0000000000001110

L'operatore XOR, se usato nel modo precedente, permette di ottenere una funzione interessante. È infatti da notare che un valore intero X ed un valore intero Y, danno come risultato, tramite la XOR, un valore intero W, allora il valore W ed il valore Y, tramite la XOR, ritornano il valore X. Ad esempio, il numero 45 ed il numero 78, con l'operatore XOR, ritornano il valore 99

```
(45)        0000000000101101
(78)        0000000001001110
            -------------------------
(99)        0000000001100011
```

ed il 99, con il 76, ritorna il 45 di partenza

```
(99)        0000000001100011
(78)        0000000001001110
            -------------------------
(45)        0000000000101101
```

Questo fatto può essere sfruttato per creare delle protezioni di dati (crittografazione), agendo sui bytes, ed usando come **chiave** il secondo numero delle operazioni precedenti; esso sarà quello tramite il quale si otterranno i dati crittografati, e sarà lo stesso che permetterà di 'decodificare' i dati codificati in precedenza.

Gli operatori che agiscono sulle stringhe, infine, sono molto semplici e si riducono all'unione (**+**) che permette di ottenere una stringa dall'unione di più stringhe e agli operatori di relazione tra stringhe (< > = >= <= <>) che permettono di testare il fatto che una stringa sia, alfabeticamente, precedente o successiva, rispetto ad un'altra (in base al codice ASCII).
Come esempio di utilizzo del primo operatore, è sufficiente questo piccolo programma

```
CLS
A$=”Quick”
B$=”Basic”
C$=A$+” “+B$
PRINT C$
END
```

mentre, per gli operatori relazionali, si deve precisare che esiste differenza tra le lettere maiuscole e quelle minuscole, dato che nel codice ASCII queste sono poste in zone diverse. Per rimediare, e quindi per ottenere risultati sempre corretti, basta usare la funzione UCASE$ **prima** della comparazione. Ad esempio, i dati definiti nelle due linee seguenti

```
A$=”Quick”
B$=”QUICK”
```

sono considerati diversi dal QB, ed una eventuale linea del tipo

PRINT A$>B$

restituirebbe il valore -1 (vero) dato che le lettere minuscole sono, nel codice ASCII, seguente le maiuscole ed hanno, quindi, codice maggiore. Ma la linea

PRINT UCASE$(A$)>UCASE$(B$)

fornisce, invece, il valore corretto.

6.2 LE STRUTTURE DI CONTROLLO

6.2.1 Uso delle strutture decisionali

Questo tipo di istruzioni permettono l'esecuzione di blocchi alternativi di programma, in base al risultato di una espressione di tipo logico-relazionale (v. 6.1.7). Di questo gruppo fanno parte

- la IF..THEN..ELSE..END IF
- la ON GOTO…
- la ON GOSUB…
- la SELECT CASE..CASE..CASE ELSE..END SELECT

La struttura IF… nella sua versione più semplice, permette di eseguire un blocco di istruzioni solo se è **vero** il risultato di una espressione logico-relazionale indicata tra le parole chiave IF e THEN. La seguente riga

mostra come sia possibile eseguire delle istruzioni solo se è verificata una espressione logico-relazionale; infatti, solo se il valore della variabile A è maggiore di quello di B, le istruzioni seguenti la THEN sono eseguite; in qualsiasi altro caso, esse sono ignorate e, comunque, l'esecuzione passa all'istruzione seguente l'IF. L'insieme di istruzioni eseguite da tale struttura, in questo caso, è limitato a quelli presenti **sulla stessa linea** in cui compare la IF; esse sono separate dal simbolo ':' e sono, quindi, di numero limitato. È possibile eseguire, in alternativa al primo gruppo di istruzioni, un **secondo** gruppo di istruzioni solo se la condizione **non** è verificata; ciò è possibile tramite l'istruzione ELSE

Ma anche in questo caso (soprattutto in questo caso), le funzionalità di tali istruzioni sono limitate dalla lunghezza della riga e dal fatto che, in questa forma, non è molto chiara la funzione svolta dalla struttura.

Il QB pone rimedio a ciò introducendo un nuovo modo di scrivere l'istruzione IF..THEN..ELSE : il blocco IF..END IF. La precedente struttura poteva quindi, essere scritta

```
IF A>B THEN        ' Inizio blocco IF con espr. logica
    PRINT A+C      ' Istruzioni eseguite solo se
    PRINT A-B      ' espr. logica è vera
ELSE               ' Clausola ELSE (opzionale)
    PRINT B-A      ' Istruzioni presenti solo se ELSE è
    PRINT A-C      ' indicata; eseguite solo se espr. logica è falsa
END IF             ' Fine blocco IF
```

Sotto questa forma, l'istruzione IF assume un'importanza fondamentale (specialmente in programmazione strutturata), diventa molto più comoda, potente e, perché no, elegante.

Ad ogni END IF indicato deve corrispondere una IF e, perché la cosa non crei confusione, si **indenta** il testo, si spostano, cioè, le istruzioni a destra quando fanno parte di un nuovo blocco. Ecco che è così possibile inserire dei blocchi IF..END IF all'interno di altri blocchi IF..END IF, in questo modo

```
IF A>B THEN --------------------> Primo blocco IF..
|   IF Z<X THEN -----------> Secondo blocco IF..
|   |       PRINT Z
|   ELSE
|   |       PRINT X
|   END IF
ELSE
|   IF T>Y THEN -----------------> Terzo blocco IF..
|   |       PRINT T
|   END IF
END IF
```

In questo caso, molto semplice in verità, i blocchi IF..END IF sono visualizzati molto chiaramente, e le linee tracciate (che non fanno parte del programma) li evidenziano.
Il QB mette a disposizione, inoltre, un'altra clausola da porsi all'interno dei blocchi IF..END IF (oltre alla ELSE) : la **ELSEIF**. Tramite tale opzione è possibile realizzare una struttura decisionale complessa; porre attenzione al fatto che con la clausola ELSEIF, **non inizia** un altro blocco IF..END IF. È così possibile scrivere delle strutture che realizzino decisioni multiple, come la seguente

```
CLS
INPUT "A : ", A
INPUT "B : ", B
PRINT
IF A>B THEN
```

```
    PRINT “A è maggiore di B”
ELSEIF A<B THEN
    PRINT “A è minore di B”
ELSE
    PRINT “A e B sono uguali”
END IF
END
```

Notare che l’ultima clausola ELSE viene eseguita solo se tutte le espressioni logico-relazionali espresse dopo la IF e le ELSEIF si sono rivelate false. Molte volte, invece di usare l’opzione ELSEIF in una struttura IF...END IF, può essere conveniente usare la struttura SELECT CASE..END SELECT di cui si parlerà in seguito.

Nei casi in cui è necessario effettuare dei salti (con istruzioni GOTO e GOSUB) in base al contenuto di una variabile numerica (situazione ricorrente nella gestione dei menu), è possibile usare l’istruzione IF, nel seguente modo

```
INPUT N
IF N=1 THEN
        GOTO Opz1
ELSEIF N=2 THEN
        GOTO Opz2
        …
ELSEIF N=n THEN
        GOTO Opzn
END IF
```

Ma per questi casi esistevano, con il BASICA ed il GW-BASIC, due strutture più efficienti, la ON GOTO… e la ON GOSUB… Anche il QB mantiene queste istruzioni, tanto che il precedente esempio può essere scritto

```
INPUT N
ON N GOTO Opz1, Opz2, …, Opzn
```

(o con la ON GOSUB se si desidera il rientro della routine), in maniera più compatta ed efficiente. Ma in QB, per aumentarne la potenza e flessibilità, è stata introdotta un’altra struttura di controllo che prevede altre possibilità: la **SELECT CASE..END SELECT**. Questa è, per la sua funzione, simile alla switch..case del linguaggio C, alla case..of del Pascal ed alla do case..endcase del Clipper, ed è usata per eseguire **qualsiasi** blocco di istruzioni in base al contenuto di una variabile secondo delle scelte multiple. Ad esempio, il programma precedente, potrebbe essere riscritto

```
INPUT N
SELECT CASE N
    CASE 1
        GOTO Opz1
    CASE 2
        GOTO Opz2
```

```
        CASE 3
            GOTO Opz3
            …
        CASE n
            GOTO Opzn
    END SELECT
```

Da notare che, in questo modo, la potenza e flessibilità della SELECT CASE, non è sfruttata al massimo. Infatti, al posto delle singole istruzioni GOTO è possibile inserire un numero elevato di istruzioni, addirittura di programmi interi, ed i controlli effettuati dalla CASE sull'espressione posta dopo la SELECT CASE, possono essere più complessi ed articolati. Ecco, ad esempio, come è possibile sfruttare tale struttura per controllare se un dato valore, rientra all'interno di un intervallo prestabilito

```
CLS
INPUT "Valore minimo intervallo : ", VMin
INPUT "Valore massimo intervallo : ", VMax
INPUT "Valore da controllare : ", V
SELECT CASE V
    CASE IS  < VMin
        PRINT "Il valore indicato e' inferiore al minimo."
    CASE IS  > VMax
        PRINT "Il valore indicato e' superiore al massimo."
    CASE IS  = VMin
        PRINT "Il valore indicato e' uguale al minimo."
    CASE IS  = VMax
        PRINT "Il valore indicato e' uguale al massimo."
    CASE ELSE
        PRINT "Il valore e' all'interno dell'intervallo."
END SELECT
```

La potenza di queste strutture deriva dal fatto che esse possono essere usate in maniera **annidata**, l'una dentro l'altra, al fine di creare un efficiente sistema di controllo dell'elaborazione dei dati. Si supponga di dovere risolvere, con tali strutture, il seguente problema: "*si deve scrivere la parte di un programma che permetta di eseguire dei salti, in base al contenuto di una variabile di controllo Jump, a tre routines diverse, il cui inizio è indicato, rispettivamente, dalle etichette Rou1, Rou2 e Rou3, se non esiste una condizione di errore (indicata dal contenuto della variabile Errore); in caso contrario, bisogna eseguire le routines DrErr1, DrErr2, o DrErr3, sempre in base al contenuto della variabile Jump, se esiste un errore di tipo 1; se invece esiste un errore di tipo 2, bisogna eseguire le routines PrErr1, PrErr2, PrErr3 a seconda del contenuto di Jump; per tutti gli altri errori, indipendentemente da Jump, bisogna eseguire la routine OthErr, ma solo se la variabile ActErr è diversa da zero; l'esecuzione del programma deve, comunque, continuare dopo l'esecuzione di una qualsiasi routine suddetta*".
Ed ecco la semplice soluzione, di un problema, a prima vista, complesso

```
SELECT CASE Errore
    CASE 0
        ON Jump GOSUB Rou1, Rou2, Rou3
```

```
        CASE 1
                ON Jump GOSUB DrErr1, DrErr2, DrErr3
        CASE 2
                ON Jump GOSUB PrErr1, PrErr2, PrErr3
        CASE ELSE
                IF ActErr THEN
                        GOSUB OthErr
                END IF
END SELECT
```

Notare i seguenti fatti

- sono state usate le ON GOSUB e non le ON GOTO per permettere il rientro e la prosecuzione del programma, come indicato dall'ultimo punto dell'esercizio;

- è stata usata la clausola CASE ELSE nella SELECT CASE per i casi di errore diversi da 1 e 2; in precedenza, erano stati affrontati i casi di 'nessun errore' (CASE 0), 'errore 1' (CASE 1) ed 'errore 2' (CASE 2);

- è stata preferita, per la IF, la forma a blocco, solo per una questione di leggibilità; era possibile usare anche il formato in un'unica linea;

- come condizione logica, nella IF, è stata specificata solo la variabile interessata; infatti, le espressioni eguali a zero sono intese false da QB, mentre, per qualsiasi altro risultato diverso da zero, l'espressione è considerata vera; è quello che serve secondo le indicazioni del penultimo punto dell'esercizio.

6.2.2 Uso delle strutture cicliche

Le strutture cicliche sono usate quando è necessario **ripetere** l'esecuzione di una determinata porzione di programma, un certo numero di volte o fino a quando non si verifica una determinata condizione. Le strutture di questo tipo che il QB mette a disposizione del programmatore, sono

- la FOR..TO..NEXT..STEP
- la WHILE..WEND
- la DO..LOOP

Mentre le prime due sono presenti anche nel BASICA e nel GW-BASIC, l'ultima è stata introdotta con il QB e rappresenta una versione più efficiente e flessibile della WHILE..WEND; per questo motivo, la WHILE..WEND è poco usata in QB. La prima di queste strutture, la FOR.., è usata quando devono essere ripetute delle istruzioni un numero di volte ben **determinato**. Essa, infatti, prevede una variabile numerica usata come contatore, che assume un valore iniziale e si incrementa di un altro valore fino ad uno finale; durante questi incrementi, vengono eseguite le istruzioni poste all'interno delle istruzioni FOR e NEXT. Ad esempio, per visualizzare i valori da 1 a 10, bastano le seguenti linee

```
FOR X=1 TO 10
       PRINT X
NEXT X
```

Considerare che, in mancanza dell'istruzione STEP per la specifica del valore che deve essere sommato al contatore ad ogni passo, esso è inteso, per default, eguale ad 1; in questo caso, la variabile X assume, dunque, valori crescenti a partire da 1 e viene stampata tramite l'istruzione PRINT compresa all'interno del ciclo; l'istruzione NEXT, incrementa la variabile X, usata come contatore, e se questa non è ancora maggiore del valore finale specificato dopo la clausola TO, permette di ripetere automaticamente il ciclo.
È possibile, precisandolo dopo l'istruzione STEP, specificare un valore del 'passo' anche decimale, in questo modo

```
FOR X=1 TO 10 STEP .05
       PRINT X
NEXT X
```

così da potere ottenere tutti i valori intermedi; per fare ciò, la variabile usata non deve essere di tipo intero, altrimenti non potrebbero essere mai trattati i decimali.
La variabile di controllo, pur potendo essere di qualsiasi tipo numerico, non può essere un elemento di un array né di tipo utente; è comunque consigliabile, quando possibile, usare come variabile una di tipo intero dato che ciò velocizza l'esecuzione del ciclo (v. 6.1.1).
Il valore del 'passo' può essere, inoltre, **negativo** (in questo caso l'opzione STEP è necessaria) così da permettere un valore di partenza della variabile di controllo maggiore del valore finale; il ciclo, in questo caso, avrà termine solo quando il valore della variabile sarà minore del valore finale.
Tenere presente che, se il valore iniziale è maggiore del finale in un ciclo a passo positivo o, il valore iniziale è minore del finale in uno a passo negativo, il ciclo stesso **non** viene eseguito neanche una volta; il controllo del programma passa, comunque, all'istruzione specificata dopo la NEXT. Ad esempio, i prossimi due esempi, mostrano quanto detto per i cicli che non possono essere eseguiti

```
FOR X=10 TO 1
       …
NEXT X

FOR X=10 TO 1 STEP -1
       …
NEXT X
```

Certe volte è necessario uscire da un ciclo FOR **prima** del suo termine ed, invece di assegnare alla variabile, in maniera deliberata, un valore che fa terminare il ciclo, o usare la poco raccomandabile istruzione GOTO, il QB disponde dell'istruzione **EXIT FOR**, appositamente studiata allo scopo. Essa, eseguita, per lo più, in base ad una condizione logica, permette di interrompere il ciclo in cui è posta ed uscire all'istruzione seguente la NEXT. Il prossimo esempio mostra come è possibile usare tale istruzione per permettere l'interruzione di un ciclo FOR..NEXT prima del suo termine, tramite l'uso di un tasto

```
FOR X=1 TO 5000
    PRINT X,
    IF INKEY$<>"" THEN
        BEEP
        EXIT FOR
    END IF
NEXT X
```

Esiste, comunque, un problema nell'uso di tale istruzione con i cicli FOR..NEXT **nidificati**, inseriti cioè, uno nell'altro. È possibile ciò nel seguente modo

```
          ----------> FOR X=1 TO 10
primo  |  secondo     -----> FOR Y=1 TO 100
ciclo  |  ciclo       |         PRINT Y
       |              -----> NEXT Y
          ----------> NEXT X
```

Notare che

- il ciclo più interno è compreso **interamente** nel più esterno; l'istruzione NEXT è eseguita, per prima, per il ciclo più interno;

- le variabili di controllo usate per i due cicli devono, chiaramente, essere **diverse** tra loro

In questo caso, una istruzione EXIT FOR posta nel ciclo più interno, fa uscire da questo ciclo ma non da quello più esterno; molte volte, invece, bisogna anche uscire da quest'ultimo e ciò è possibile, usando un tasto, con questa semplice tecnica di programmazione

```
EX=0
FOR X=1 TO 10
    FOR Y=1 TO 100
        PRINT Y
        IF INKEY$<>"" THEN
            BEEP
            EX=-1
            EXIT FOR
        END IF
    NEXT Y
    IF EX THEN
        EXIT FOR
    END IF
NEXT X
```

che usa una variabile di comodo (EX) per indicare l'uscita dal ciclo interno a quello più esterno.

La struttura WHILE..WEND, eredità del BASICA e del GW-BASIC, è indicata per ripetere un certo gruppo di istruzioni, non un numero di volte definito, ma finché una condizione logica è vera. Essa prevede, infatti, la specifica di tale condizione di seguito all'struzione WHILE e, se tale condizione viene giudicata vera, il ciclo viene eseguito. È compito delle istruzioni interne al ciclo, o di una evento esterno ed asincrono, intervenire sulla condizione logica per modificarne il valore al fine di terminare il ciclo. Questo può, in maniera deliberata, essere fatto in modo da non avere termine, specificando una costante sempre vera, come condizione logica.
Il seguente esempio mostra l'uso della WHILE per visualizzare una frase lampeggiante in due colori diversi ed attendere un tasto

```
DEFINT A-Z
CONST Avv = "Premere un tasto …"
CONST Ritardo = .3
CLS
WHILE INKEY$=""
        COLOR 7
        LOCATE 10, 10
        PRINT Avv
        Ora!=TIMER
        WHILE TIMER-Ora! < Ritardo
        WEND
        COLOR 7
        LOCATE 10, 10
        PRINT Avv
        Ora!=TIMER
        WHILE TIMER-Ora! < Ritardo
        WEND
WEND
COLOR 15
LOCATE 10, 10
PRINT "Ok."
END
```

Notare che,

- è possibile specificare, nell'espressione logica dopo la WHILE, delle funzioni (INKEY$ e TIMER) che ritornano un valore sempre diverso, dato che esse vengono valutate **ogni volta** che inizia il ciclo;

- è necessario, in questo esempio, specificare che la variabile Ora! sia in singola precisione dato che il Ritardo per il lampeggio, indicato in secondi, non è intero;

- le WHILE..WEND, come le altre strutture, possono essere **nidificate** ed ad ogni WEND è associata la WHILE più interna;

- la valutazione ed il controllo dell'espressione logica viene fatto **all'inizio** del ciclo WHILE..WEND; così, se la condizione logica è, in partenza, **falsa**, il ciclo non viene eseguito neanche una volta.

Dato che la rilevazione del tasto è fatta dalla WHILE più esterna, potrebbe essere avvertito un ritardo tra l'uso del tasto stesso e la fine del programma; questo effetto potrebbe essere più grande quando si sceglie un ritardo maggiore; infatti, il tasto potrebbe essere stato premuto, nel caso peggiore, all'inizio del primo ciclo WHILE interno e, dato che, come detto, la condizione viene valutata all'inizio del ciclo, si dovrebbe attendere il termine dei due cicli più interni (due volte il tempo di ritardo) per uscire dal più esterno. Purtroppo non esiste un diverso metodo per uscire dal ciclo WHILE..WEND, ma, il QB offre la struttura DO..LOOP che pone rimedio a problemi simili a quello presentato.
Il programma precedente, riscritto usando la DO..LOOP, è il seguente

```
DEFINT A-Z
CONST Vero = -1, Falso = NOT Vero
CONST Avv = “Premere un tasto …”
CONST Ritardo = .3
CLS
EX=Falso
DO WHILE Vero
        COLOR 15
        LOCATE 10, 10
        PRINT Avv
        Ora!=TIMER
        DO WHILE TIMER-Ora! < Ritardo
                IF INKEY$<>”” THEN
                        EX = Vero
                        EXIT DO
                END IF
        LOOP
        COLOR 0
        LOCATE 10, 10
        PRINT Avv
        Ora!=TIMER
        DO WHILE TIMER-Ora! < Ritardo
                IF INKEY$<>”” THEN
                        EX = Vero
                        EXIT DO
                END IF
        LOOP
        IF EX THEN
                EXIT DO
        END IF
LOOP
COLOR 15
LOCATE 10, 10
PRINT “Ok.”
```

```
END
```

In questo caso

- è possibile controllare l'uso del tasto dentro i due cicli DO..LOOP più interni in modo da terminarli, con l'apposita istruzione **EXIT DO**, appena ciò avviene; l'istruzione EXIT DO, in modo simile alla EXIT FOR per il ciclo FOR..NEXT, termina l'esecuzione del ciclo DO..LOOP in cui viene eseguita, a prescindere dal valore della condizione logica di controllo;

- notare che, come nel caso di FOR..NEXT nidificate, è possibile **nidificare** anche le strutture DO..LOOP e, per permettere il termine del ciclo più esterno, si usa la solita variabile di controllo EX;

- la condizione di controllo del ciclo più esterno è, in questo caso, una costante sempre vera; per questo motivo esso non ha termine tranne che con l'esecuzione di una EXIT DO al suo interno; ciò avviene solo se la variabile EX diventa vera, e cioè, solo se si preme un tasto;

- non esiste ritardo apprezzabile tra l'uso del tasto e la fine del programma dato che i cicli di ritardo sono subito interrotti dalle istruzioni EXIT DO poste al loro interno.

In questo esempio, è stata usata la clausola **WHILE** subito dopo l'istruzione DO in tutti e tre i cicli; essa prevede che il ciclo venga ripetuto finché la condizione specificata è **vera**; ma la struttura DO..LOOP prevede, in maniera molto flessibile, anche l'uso della clausola **UNTIL** tramite la quale è possibile ripetere il ciclo fino a quando la condizione logica è **falsa**. Le seguenti righe dell'esempio possono essere modificate, quindi, nel modo seguente, senza pregiudicarne il funzionamento

```
…
DO UNTIL Falso
…
     DO UNTIL TIMER - Ora! > Ritardo
     …
     DO UNTIL TIMER - Ora! > Ritardo
     …
```

(notare che, usando UNTIL al posto della WHILE, si è provveduto a modificare il valore della costante, da Vero a Falso, nel primo ciclo, e l'operatore di relazione, da minore a maggiore, negli altri due cicli).

Ma la DO..LOOP prevede ancora di più; la condizione logica può essere specificata, sempre usando le clausole WHILE o UNTIL, anche alla **fine** della struttura, dopo l'istruzione LOOP. Questo può essere utile quando la valutazione ed il controllo dell'espressione logica deve essere fatto alla fine del ciclo, per esigenze di funzionamento del programma. Il precedente programma d'esempio, funziona anche in questa maniera e si può fare quindi la prova a porre dopo l'istruzione LOOP, quanto specificato in precedenza, dopo la DO.

Si abbia il seguente problema: "*scrivere un programma che emetta un segnale acustico allo scadere di ogni secondo e che termini usando un tasto; provvedere perché* ***almeno*** *il primo segnale venga, in ogni caso, emesso*".
Esso può essere, comodamente risolto con la struttura DO..LOOP in cui la condizione è controllata alla fine della stessa; questo fatto assicurerà l'esecuzione del ciclo almeno una volta con la conseguente emissione acustica

```
A$=""
DO
        BEEP
        Ora!=TIMER
        DO WHILE TIMER - Ora! < 1  AND A$=""
                A$=INKEY$
        LOOP
LOOP WHILE A$=""
END
```

La variabile A$ viene usata, in questo esempio, per tenere traccia del tasto premuto quando viene eseguito il ciclo più interno al fine di porre termine all'esterno; è un altro modo, insieme a quello già visto, in cui si utilizza la variabile EX, per terminare il ciclo più esterno quando termina il più interno, ma questa volta, senza usare la EXIT DO.

6.3 IL TRATTAMENTO DELLE STRINGHE

6.3.1 Tipi di stringhe

Il Quick Basic è capace di gestire due tipi di dati alfanumerici: le stringhe **a lunghezza variabile** e quelle **a lunghezza fissa**. La lunghezza delle stringhe è, comunque, rappresentata dal numero di caratteri in esse contenute, che può variare da 0 (nessun carattere nella stringa, **stringa vuota**) fino a 32767 (massimo numero di caratteri consentiti in una stringa). La funzione LEN può essere usata per ricavare tale valore da una qualsiasi stringa.
Di altre caratteristiche delle stringhe, a lunghezza fissa e variabile, si è già detto all'inizio del capitolo (v. 6.1.1); nelle pagine seguenti si cercherà di dimostrare come le stringhe possano essere usate in maniera efficiente e soddisfacente.

6.3.2 Unione e comparazione di stringhe

Le stringhe possono essere combinate tra loro, in maniera da ottenerne una sola; ciò è possibile tramite l'operatore di unione delle stringhe a cui corrisponde il simbolo +
Definite più stringhe ed assegnato loro un contenuto

```
DIM S1 AS STRING, S2 AS STRING
DIM S3 AS STRING, S4 AS STRING
S1="Quick Basic"
S2="Tecniche di "
S3="programmazione avanzata."
```

è possibile assegnare ad un'altra stringa l'unione delle tre stringhe già usate, nel seguente modo

```
S4=S1+S2+S3
```

Per controllare quanto fatto, eseguire le seguenti linee

```
CLS
PRINT S4
PRINT LEN(S1), LEN(S2), LEN(S3)
PRINT LEN(S4)
END
```

Notare che è stato lasciato, appositamente, uno spazio alla fine della prima e seconda stringa, per evitare che, dopo l'unione, esse risultassero attaccate; l'operatore +, infatti, unisce le stringhe così come sono, senza preoccuparsi di porre spazi o altri caratteri dove servono.
È possibile operare su una unione di stringhe, come se fosse, effettivamente, una stringa unica, preoccupandosi però, che la sua lunghezza complessiva non superi il numero di 32767 caratteri o quello stabilito nella dichiarazione, per le stringhe a lunghezza fissa; mentre, però, nel primo caso il QB emette un errore, nel secondo i dati in più vengono persi e la stringa viene troncata al numero di caratteri dichiarati.

Nel comparare le stringhe, bisogna porre molta attenzione al fatto che, per le lettere, esiste un piccolo problema: le minuscole e le maiuscole sono considerate, a tutti gli effetti, come simboli diversi tra loro, anche se si riferiscono alla stessa lettera; questo perché, secondo il codice ASCII, alle maiuscole viene assegnato il codice compreso tra 65 e 90 e alle minuscole il codice compreso tra 97 e 123. Per effettuare una comparazione corretta, quindi, è necessario convertire prima le stringhe tutte in maiuscolo o tutte in minuscolo, al fine di ottenere risultati coerenti. Le funzione LCASE$ ed UCASE$ sono adibite allo scopo ed il loro corretto uso è dimostrato nella riga seguente

```
IF UCASE$(A$)=UCASE$(X$) THEN
```

Inoltre, a volte, possono esistere degli spazi, iniziali o finali, non desiderati, sia per questione di confronto che per la visualizzazione o stampa; il QB mette a disposizione altre due funzioni (da usare in particolar modo trattando i file random), la LTRIM$ e la RTRIM$ che ritornano la stringa di partenza, rispettivamente, senza gli spazi iniziali e finali. Per fare in modo di togliere tutti gli spazi, considerare il seguente esempio

```
CLS
A$="   Prova con le stringhe   "
PRINT RTRIM$(LTRIM$(A$))
```

Per dimostrare il funzionamento di altre funzioni che operano sulle stringhe (LEFT$, RIGHT$, MID$, LEN), considerare il seguente programma di esempio; seppure un po' più complesso, esso è strutturato in maniera semplice e svolge le sue funzioni in maniera ottima; il programma principale è costituito dalle funzioni NumToLet e NtlBlk3 che convertono un numero espresso in

cifre nell’equivalente stringa di lettere; il Main Module è inserito solo per provare le funzioni (la NumToLet è inserita nella libreria BPLUS e commentata in seguito)

```
' (Main Module)
DECLARE FUNCTION NumToLet$(N AS DOUBLE)
DECLARE FUNCTION NtlBlk3$(VN AS STRING)
DO
        CLS
        INPUT "Numero : ", Num#
        Num#=INT(Num#)
        CLS
        PRINT Num#, NumToLet(Num#)
LOOP WHILE Num# > 0
END
DEFINT A-Z
FUNCTION NtlBlk3$(PN AS STRING)
        DIM Res AS STRING
        DIM TmpRes AS STRING
        DIM NtlA(0 TO 19) AS STRING
        DIM NtlB(2 TO 9) AS STRING
        DIM NtlC(1 TO 1) AS STRING
        NtlA(0)="zero"
        NtlA(1)="uno" : NtlA(2)="due" : NtlA(3)="tre"
        NtlA(4)="quattro" : NtlA(5)="cinque"
        NtlA(6)="sei" : NtlA(7)="sette" : NtlA(8)="otto"
        NtlA(9)="nove" : NtlA(10)="dieci"
        NtlA(11)="undici" : NtlA(12)="dodici"
        NtlA(13)="tredici" : NtlA(14)="quattordici"
        NtlA(15)="quindici" : NtlA(16)="sedici"
        NtlA(17)="diciassette" : NtlA(18)="diciotto"
        NtlA(19)="diciannove" : NtlB(2)="venti"
        NtlB(3)="trenta" : NtlB(4)="quaranta"
        NtlB(5)="cinquanta" : NtlB(6)="sessanta"
        NtlB(7)="settanta" : NtlB(8)="ottanta"
        NtlB(9)="novanta" : NtlC(1)="cento"
        VN=VAL(PN)
        IF VN>99 THEN
                CN=VAL(LEFT$(PN , 1))
                IF CN=1 THEN
                        Res = NtlC(1)
                ELSE
                        Res = NtlA(VAL(LEFT$(PN, 1))) + NtlC(1)
                END IF
                VN=VAL(MID$(PN, 2))
        END IF
        IF VN>19 THEN
                TmpRes=NtlB(VAL(MID$(PN, 2, 1)))
                IF VAL(RIGHT$(PN, 1)) > 0 THEN
```

```
				UN=VAL(RIGHT$(PN , 1))
				IF UN=1 OR UN=8 THEN
					TmpRes=LEFT$(TmpRes, LEN(TmpRes)-1)
				END IF
				TmpRes= TmpRes+NtlA(UN)
			END IF
			Res= Res+TmpRes
		ELSE
			IF LEN(Res)>0 AND VN<>0 THEN
				Res= Res+NtlA(VN)
			ELSE
				IF LEN(Res)=0 THEN
					Res=NtlA(VN)
				END IF
			END IF
		END IF
		NtlBlk3$=Res
	END FUNCTION
	FUNCTION NumToLet$(N AS DOUBLE)
		DIM Res AS STRING
		DIM TmpRes AS STRING
		DIM PN(1 TO 4) AS STRING
		VN$=MID$(STR$(N#), 2)
		PNX=1
		DO WHILE LEN(VN$)>0
			PN(PNX)=RIGHT$(VN$, 3)
			PN(PNX)=STRING$(3-LEN(PN(PNX)), "0") + PN(PNX)
			DF = LEN(VN$)-LEN(PN(PNX))
			IF DF<0 THEN
				DF=0
			END IF
			VN$=LEFT$(VN$, DF)
			PNX=PNX+1
		LOOP
		FOR PNX=1 TO 4
			IF LEN(PN(PNX)) = 0 THEN
				EXIT FOR
			END IF
			SELECT CASE PNX
				CASE 1
					Res=NtlBlk3(PN(PNX))
				CASE 2
					TmpRes= NtlBlk3(PN(PNX))
					IF VAL(PN(PNX)) = 1 THEN
						TmpRes = "mille"
					ELSE
						IF TmpRes="zero" THEN
							TmpRes=""
```

```
                        ELSE
                            TmpRes=TmpRes + "mila"
                        END IF
                    END IF
                    IF Res="zero" THEN
                        Res=""
                    END IF
                    Res=TmpRes+Res
                CASE 3
                    TmpRes=NtlBlk3(PN(PNX))
                    IF VAL(PN(PNX)) = 1 THEN
                        TmpRes="unmilione"
                    ELSE
                        TmpRes=TmpRes + "milionI"
                    END IF
                Res=TmpRes+Res
                CASE 4
                    TmpRes=NtlBlk3(PN(PNX))
                    IF VAL(PN(PNX)) = 1 THEN
                        TmpRes="unmiliardo"
                    ELSE
                        TmpRes=TmpRes + "miliardi"
                    END IF
                Res=TmpRes+Res
            END SELECT
        NEXT PNX
        NumToLet$=UCASE$(LEFT$(Res, 1)) + MID$(Res, 2)
    END FUNCTION
```

6.3.3 Trattamento di parti di stringhe

È possibile gestire una stringa anche nelle sue parti, tramite funzioni che il Quick Basic mette a disposizione allo scopo. Esse sono 5 e, precisamente,

- **LEFT$**
- **RIGHT$**
- **MID$**
- **LTRIM$**
- **RTRIM$**

Le prime tre funzioni servono a prelevare da una stringa una sua parte posta, rispettivamente, a sinistra, a destra e nel mezzo della stessa. Le funzioni LEFT$ e RIGHT$ ammettono due soli parametri, la stringa da trattare ed il numero di caratteri da prelevare. Se si avesse, ad esempio, la seguente stringa posta all'interno della variabile X$

```
X$="Questa e' una prova."
```

la seguente riga

PRINT LEFT$(X$, 6)

indicherebbe che bisogna visualizzare i primi 6 caratteri, presi ad iniziare da sinistra, della stringa X$; in questo caso sarebbe quindi visualizzata, solamente, la parola

Questa

Il seguente schema mostra come la funzione LEFT$ prelevi, nell'esempio, la parola da visualizzare

Stringa ---> Questa e' una prova.
Primi 6 caratteri da sinistra ---> 123456

Allo stesso modo, la funzione RIGHT$, preleva un determinato numero di caratteri da una stringa, ma a partire da destra. La seguente riga, ad esempio

PRINT RIGHT$(X$, 6)

visualizzarebbe i caratteri

prova.

in quanto essi sono i primi sei caratteri, a partire da destra, contenuti nella stringa indicata come primo parametro

Stringa ---> Questa e' una prova.
Primi 6 caratteri da destra ---> 123456

La funzione MID$, infine, è usata quando bisogna estrarre una parte di una stringa che si trova nel mezzo e, per fare ciò, è necessario indicare, dopo la stringa da cui si deve estrarre una parte, il numero del carattere, a partire da sinistra, da cui inizia il blocco da prelevare, ed il numero di caratteri che fanno parte di tale blocco. Ad esempio, se dalla seguente stringa

IND$ = “Via Roma, 42 - 90100 - PALERMO"

si volesse estrarre solamente il CAP per visualizzarlo, si dovrebbe usare la seguente riga

PRINT MID$(X$, 16, 5)

in cui viene indicato, dopo il nome della stringa da trattare, che il blocco inizia ad sedicesimo carattere e si estende per 5 caratteri. Il seguente schema mostra come opera, in questo caso, la funzione MID$

Stringa -----> Via Roma, 42 - 90100 - PALERMO
Questo è il sedicesimo 000000000111111
carattere da sinistra -----> 1234567890123456
Il dato è lungo 5 caratteri -----> 12345

Il terzo parametro della funzione MID$ (l'estensione del blocco da prelevare) è opzionale, e quando non è specificato permette di estrarre dalla stringa **tutta la rimanente parte** alla destra del carattere specificato come inizio. Nel caso precedente, quindi, la riga

PRINT MID$(X$, 16)

avrebbe permesso di visualizzare i seguenti dati

90100 - PALERMO

Fare attenzione a non confondere la **funzione** MID$ con l'**istruzione** MID$; mentre la prima, si è visto, che è utile a prelevare parti di una stringa, la seconda serve a modificare parti di una stringa senza doverla riassegnare. Il funzionamento di tale istruzione è descritto in seguito.
Notare che è possibile simulare la funzione MID$ usand, contemporaneamente, le funzioni LEFT$ e RIGHT$. Infatti, come nell'esempio in cui si prelevava il Cap dalla stringa di indirizzo precedente, la riga

PRINT LEFT$(RIGHT$(X$, 15), 5)

dà gli stessi risultati, e lo schema seguente mostra come essa opera

Stringa	**----->**	Via Roma, 42 - 90100 - PALERMO
Primi 15 caratteri		000000000111111
da destra	**----->**	1234567890123456
Primi 5 caratteri da sinistra	**----->**	12345

Notare che, in questo caso, la funzione RIGHT$ agisce sull'intera stringa X$, mentre, la funzione LEFT$ opera soltanto sulla stringa fornita come risultato dalla funzione RIGHT$. Dato che l'espressione viene calcolata a partire dalle funzioni più interne verso le più esterne, il risultato è sicuramente corretto.

Le funzioni LTRIM$ e RTRIM$ sono utili quando è necessario, come nel caso del trattamento deli file random, operare su dati alfanumerici che iniziano o terminano con degli spazi indesiderati. La funzione LTRIM$, infatti, provvede ad eliminare tali spazi alla **sinistra** di una stringa, mentre, la RTRIM$, opera in modo similare ma alla **destra**. Il seguente esempio mostra l'utilizzo delle due funzioni

```
X1$="     Essere o non essere"
X2$="Essere o non essere       "
IF X1$<>X2$ THEN
        PRINT "Le stringhe sono diverse"
END IF
IF LTRIM$(X1$)=RTRIM$(X2$) THEN
        PRINT "Le stringhe sono uguali"
END IF
END
```

Questo programma infatti, mostra che le due stringhe sono considerate differenti ma, con l'uso delle funzioni che eliminano gli spazi superflui, esse sono considerate eguali.

Notare che, per eliminare gli spazi iniziali e finali di una stringa, è sufficiente usare contemporaneamente le due funzioni, come nel seguente esempio

```
X$="    Essere o non essere    "
PRINT LTRIM$(RTRIM$(X$))
```

6.4 USARE I FILE

6.4.1 La gestione dei file

Sapere gestire bene i file, indubbiamente, è per il programmatore, in qualsiasi linguaggio di programmazione, un fatto fondamentale. La corretta gestione dei file, insieme alla padronanza delle istruzioni e delle funzioni messe a disposizione dal linguaggio di programmazione e delle tecniche per la risoluzione dei problemi, sono le caratteristiche che differenziano un buon programmatore da un programmatore inesperto.

Il QB mette a disposizione del programmatore, ben tre tipologie di file trattabili, a differenza delle precedenti versioni di BASIC che ne offrivano solo due. Permangono, infatti, i file **sequenziali** ed i file **random**; in più, possono essere trattati i file che il QB chiama **binari**. Data la complessità dell'argomento, è bene vedere in dettaglio il trattamento dei tre tipi di file, con i soliti esempi pratici.

I file sequenziali

È il tipo più semplice di file che il QB mette a disposizione dell'utente; esso è organizzato in maniera semplice, a righe di lunghezza variabile, che possono contenere diversi dati. Un modo per capire i file sequenziali è offerto dall'analogia che si può fare con le cassette audio su cui l'informazione sonora è memorizzata in maniera continua (sequenziale) su una traccia del nastro in modo che abbia un inizio ed una fine. Allo stesso modo, i dati memorizzati in un file sequenziale, sono conservati, in maniera continua, separati con dei caratteri speciali, ed hanno un inizio e una fine. Caratteristica essenziale che differenzia tali file è appunto il **metodo di accesso sequenziale**, unico metodo usabile per scrivere o leggere dei dati. Questo metodo obbliga a trattare tutti i dati precedenti a quello che si vuole gestire e non permette quindi un accesso 'diretto' al dato che interessa; cosicché, se si vuole leggere il quattordicesimo dato scritto su un file sequenziale, bisogna, necessariamente, iniziare a leggere dal primo, i primi tredici, ignorandoli, per arrivare al dato che interessa; nella scrittura, inoltre, per modificare, ad esempio, il ventesimo dato, è necessario **riscrivere** i primi 19 su un secondo file, scrivere il dato modificato e, in seguito, quelli seguenti il ventesimo dato fino alla fine del primo file; a questo punto basta cancellare il primo file e considerare il secondo file così ottenuto come l'originale.

È chiaro che, dati questi obblighi, la gestione di una grande quantità di dati che variano in maniera veloce, è quasi impossibile con tale tipo di file, il cui uso è invece riservato alla conservazione su disco di piccole quantità di dati che devono essere trattati in maniera limitata.

È comunque un tipo di file molto usato da chi inizia a programmare, proprio per la sua estrema semplicità d'uso; sono poche, infatti, le istruzioni per permetterne l'uso e sono di facile comprensione.

Come per qualsiasi tipo di file gestibile con il Quick Basic, per i sequenziali esistono delle istruzioni e funzioni adatte a realizzare le seguenti operazioni

- apertura del flusso
- lettura dati
- scrittura dati
- chiusura del flusso

Ogni file deve essere **aperto** prima di poterlo trattare e ciò è possibile tramite l'istruzione **OPEN**. Per i sequenziali l'istruzione OPEN prevede la specifica del nome del file, del numero del canale associato e del codice di accesso. Quest'ultimo può essere uno dei tre seguenti

I Input (accesso in lettura)
O Output (accesso in scrittura)
A Append (accesso in scrittura per aggiungere)

Per il primo tipo di accesso è necessaria la preventiva presenza del file da leggere, pena l'emissione di un errore; la modalità di output, invece, azzera un file già esistente mentre lo crea nuovo se non esiste; come la precedente, la modalità append è usata per scrivere dati ma, a differenza della output, essa serve a non distruggere dati già esistenti in un file creato in precedenza; con la modalità append dei nuovi dati possono essere 'accodati' a quelli vecchi già contenuti nel file; se il file non esiste, la modalità append crea il file nuovo come fa la output.

Per potere scrivere su file sequenziale esistono, essenzialmente, due istruzioni, la PRINT # e la WRITE # che hanno un comportamento leggermente diverso. Mentre la prima, infatti, non pone alcun separatore tra i dati, a meno di usarne una per ogni dato, la seconda scrive automaticamente dei caratteri di separazione necessari in fase di lettura per il sicuro riconoscimento dei dati utente.

La lettura dei dati sequenziali è fatta, per lo più, con l'apposita istruzione INPUT # ma può essere usata anche la LINE INPUT #.

Per ogni dato da leggere dal file deve essere specificata una variabile del tipo adeguato e la fine del file va controllata per evitare errori durante la lettura. A questo proposito è necessario precisare che tale controllo va sempre fatto **prima** di usare un'istruzione di lettura da file sequenziale perché la lettura di dati oltre la fine del file è impossibile. Molto spesso si incorre in questo errore solo perché il controllo della fine del file viene fatto dopo una lettura.

Il programma tipo per la scrittura di un file sequenziale, è il seguente

```
CLS
OPEN "Prova" FOR OUTPUT AS #1
A$=""
DO WHILE A$<>"Fine"
```

```
        INPUT “Dato “, D$
        PRINT #1, D$
LOOP
CLOSE #1
END
```

Da notare che la chiusura del file (istruzione CLOSE #1) deve essere eseguita perché essa ha, anche, la funzione di scrivere tutti i dati eventualmente presenti nel buffer per poterlo liberare in seguito.
La lettura di un file sequenziale tipica, è invece la seguente

```
CLS
OPEN “Prova” FOR INPUT AS #1
A$=””
DO WHILE NOT EOF(1)
        INPUT #1, D$
        PRINT D$
LOOP
CLOSE #1
END
```

È evidente il fatto che il controllo della fine del file (condizione NOT EOF(1)) viene fatta **prima** della lettura dello stesso (istruzione INPUT #) per garantire una perfetta scansione del file in questione.

L’uso dell’istruzione LINE INPUT # per leggere dati da un file sequenziale, può rendersi necessaria quando in questi dati sono presenti dei caratteri (come la virgola) che vengono interpretati dall’istruzione INPUT # come separatori; è il caso in cui si debbano conservare degli indirizzi del tipo

Via Roma, 22
Via Lazio, 12
Via Lombardia, 99

La LINE INPUT# infatti, legge correttamente dall’inizio alla fine della riga al contrario della INPUT # che considera, per ogni riga, due dati separati.

I file random

A differenza dei sequenziali, i file random presentano un record con lunghezza fissa senza alcun carattere separatore tra due records diversi. È per tali caratteristiche che il sistema può posizionarsi molto velocemente su un singolo record e modificarlo senza occuparsi degli altri records eventualmente presenti all’interno del file.
Come per i sequenziali, i file random devono essere aperti, sempre con l’istruzione OPEN, in cui si può (ma è opzionale) specificare la lunghezza del record (in caratteri senza separatori). Ad esempio, per aprire un file denominato PROVA sul canale 1, con un record lungo 52 caratteri, è sufficiente la seguente linea

OPEN “PROVA” FOR RANDOM AS #1 LEN=52

o, in alternativa, quest’altra

OPEN “R”, #1, “PROVA”, 52

Subito dopo la OPEN è necessario specificare la struttura del record, con i nomi dei campi e le loro lunghezze. Ciò si ottiene con l’istruzione FIELD, ad esempio, nel seguente modo

FIELD #1, 20 AS NOME$, 20 AS COGNOME$, 12 AS TELEFONO$

ponendo attenzione al fatto che il totale delle lunghezze dei campi non deve superare la lunghezza del record specificata nella OPEN. È comunque possibile, con il Quick Basic, provvedere alla definizione dei campi del file random, con l’istruzione TYPE..END TYPE ed i dati utente. Per l’uso di tali istruzioni si rimanda, comunque, alla descrizione della **TYPE..END TYPE** e delle funzioni **CV..** e **MK..** in cui vengono chiariti tali concetti.

Se si usa l’istruzione FIELD #, l’input da tastiera dei dati da conservare su disco va fatto su variabili con nome **diverso** da quello usato per le variabili definite nel buffer (quelle definite nella FIELD #). Questo fatto è necessario perché, per accedere a tale buffer, sono usate due istruzioni apposite, la **LSET** e la **RSET**. Quest’ultime istruzioni **non sono necessarie** se si usa la struttura TYPE..END TYPE.

Dopo avere preparato i dati nel buffer con delle istruzioni LSET, per potere scrivere fisicamente si disco i dati è usata l’istruzione PUT # con la quale si specifica il numero di canale ed il numero di record su cui si vuole operare.

In maniera similare, per leggere i dati da un determinato record, è sufficiente leggere l’istruzione GET # che trasferisce i dati dal disco alle variabili definite nel buffer. Il loro contenuto, può essere direttamente usato in visualizzazione, calcolo o stampa, o secondo altre necessità.

L’istruzione CLOSE # è necessaria anche con i file random, per garantire che tutte le istruzioni PUT # scrivano effettivamente su disco tutti i dati ad esse collegati.

Per definizione, con i file random, è possibile specificare, **in qualsiasi ordine**, i record da scrivere o leggere senza una sequenza predefinita.

È possibile usare una chiave alfanumerica associandola ad un record di un file random; memorizzando in un file sequenziale (o random) una lista sempre ordinata di tali chiavi, è possibile accedere in maniera veloce a qualsiasi dato ricercato (**indicizzazione**). Dopo aver trattato i file binari, si potrà esaminare un esempio di tale metodo.

I file binari

Quick Basic permette l'uso dei file binari e quindi la lettura e la scrittura diretta, byte per byte, di un qualsiasi file gestito dal DOS, anche non ASCII. Con questo tipo di file è possibile scrivere o leggere uno solo dei caratteri componenti il file senza preoccuparsi di caratteri delimitatori o speciali; per tale motivo può essere aperto in tale modo, anche un file creato come sequenziale o random per esaminarlo e modificarne uno o più bytes. Tramite i file binari è possibile, adesso, scrivere dei programmi che permettano la copia, lo spostamento ed il confronto di qualsiasi file del DOS e risulta così semplice scrivere utility di sistema che utilizzino i file di MS-DOS.

L'istruzione di apertura di un file binario, è la seguente

OPEN "Prova" FOR BINARY AS #1

oppure

OPEN "B", #1, "Prova"

Appena aperto un file in modalità binaria, viene usato da Quick Basic un valore, detto puntatore al file, il cui contenuto indica il prossimo byte che può essere scritto o letto; esso è naturalmente predisposto per trattare il primo byte subito dopo l'esecuzione dell'istruzione OPEN.

La lettura e la scrittura di dati con questo tipo di file, avviene, come per i file random, con le istruzioni GET e PUT; in questo caso, però, in tali istruzioni non viene specificato il numero di record ma, la **posizione** e una **variabile** (di qualsiasi tipo) il cui contenuto deve essere scritto o che rappresenta il buffer di lettura.
Ad esempio, per leggere i primi 30 bytes di un file denominato PROVA, può essere utile il seguente programma

```
CLS
X$=SPACE$(30)
OPEN "PROVA" FOR BINARY AS #1
GET #1 , , X$
PRINT X$
CLOSE #1
END
```

Da notare che

- la larghezza della variabile X$ (che costituisce il buffer di input) è l'unico parametro che definisce il numero di caratteri da leggere;

- il fatto di avere posto due virgole prima del nome della variabile nell'istruzione GET #, non è un errore; questa scrittura è usata per permettere la lettura dei caratteri del file ad iniziare dal carattere specificato dal puntatore; tenere presente che questo è posto ad 1 all'apertura del file e che, quindi, in questo caso, vengono letti i primi 30 caratteri del file PROVA.

Se si volessero trattare dei caratteri che stanno in mezzo al file, ad esempio i 40 caratteri posti a partire dal 400-mo carattere, basterebbe definire la variabile della lunghezza adeguata con l'istruzione

X$=SPACE$(40)

ed usare la seguente riga per la lettura dei dati

GET #1, 400, X$

In maniera similare, con l'istruzione PUT, si opera per la scrittura dei dati; essi saranno predisposti all'interno della variabile usata nella stessa PUT e saranno scritti tutti a partire da quello indicato o, in sua assenza, da quello attualmente indicato dal puntatore.

Per spostare il puntatore senza effettuare alcuna operazione di ingresso / uscita (senza cioè usare la GET o la PUT), è disponibile l'**istruzione SEEK** che, insieme alla **funzione SEEK** permette di gestire il puntatore.

Anche con questi file è necessario chiudere il file (con l'istruzione CLOSE #) per garantire il completo e corretto trasferimento di tutti i dati scritti con le istruzioni PUT.

File di tipo BLOAD / BSAVE

Un particolare tipo di file gestito dal Quick Basic (ma non solo da questo compilatore Basic), è quello creato tramite l'istruzione **BSAVE** e che è possibile leggere con l'istruzione **BLOAD**.
L'istruzione BSAVE serve, essenzialmente, a registrare su disco l'immagine di una zona di memoria di cui vengono specificati il segmento e l'offset iniziali e la lunghezza in bytes. Il file così creato è, praticamente, un file binario, ma dispone di una 'testata' di 7 bytes tali che ne permettono il riconoscimento da parte della BLOAD. La funzione dei bytes presenti in questa testata è stata esaminata nella descrizione della stessa istruzione (v. 5.1.1); sempre in tale contesto è stato visto qualche esempio d'uso di tali istruzioni, usate per registrare e rileggere interi arrays presenti in memoria.

Il seguente esempio, più complesso di quelli visti fino ad adesso, è abbastanza completo e mostra l'uso dei file random e di quelli di tipo BLOAD/BSAVE per realizzare una 'agenda telefonica indicizzata'.
I file usati nell'esempio sono due, AGE.DAT e AGE.IDX, che sono, rispettivamente, il file random che contiene i dati dell'agenda ed il file binario, di tipo BLOAD/BSAVE, che contiene le chiavi ordinate alfabeticamente. Ad ogni immissione di nuovi dati o cancellazione di vecchi, viene aggiornato tale file in maniera che, la routine di ricerca binaria riesca a funzionare al meglio

```
' $DYNAMIC
DECLARE SUB AddPersona ()
DECLARE SUB Cancel ()
DECLARE SUB InpDat ()
DECLARE SUB LstElenco ()
DECLARE SUB Modify ()
```

```
DECLARE SUB NumEls ()
DECLARE SUB Pack ()
DECLARE SUB QSortIndex (Sinistra%, Destra$%)
DECLARE SUB ReIndex (FWait%)
DECLARE SUB ReadIndex (Rtyp%)
DECLARE SUB Recup ()
DECLARE SUB Selec ()
DECLARE SUB VisDat (Stri$)
DECLARE SUB VisNom (Nelem%, Nfile%)
DECLARE SUB WriteIndex ()
DEFINT A-Z
CLEAR , , 512
CONST Falso = 0, Vero = NOT Falso
CONST Modif = Vero, Normal = Falso
CONST ReadFirst = Vero, ReRead = Falso
CONST YesWait = Vero, NoWait = Falso
TYPE UnaPersona
        Cognome AS STRING * 25
        Nome AS STRING * 25
        Indirizzo AS STRING * 30
        Localita AS STRING * 25
        Cap AS STRING * 5
        Provincia AS STRING * 2
        TelefonoCasa AS STRING * 12
        TelefonoLavoro AS STRING * 12
Deleted AS STRING * 1
END TYPE
TYPE Idx
        Cognome AS STRING * 25
        Nome AS STRING * 25
        Record AS INTEGER
END TYPE
DIM SHARED Nelements AS INTEGER
DIM SHARED CurrEl AS INTEGER
DIM SHARED Persona AS UnaPersona
REDIM SHARED Index(1 TO 1) AS Idx
Nelements=0
CurrEl=1
ReadIndex ReadFirst
DO
        CLS
        LOCATE 5, 30
        PRINT "Agenda Telefonica"
        LOCATE 7, 20
        PRINT "1 - Inserimento"
        LOCATE , 20
        PRINT "2- Selezione"
        LOCATE , 20
```

```
            PRINT “3 - Modifica”
            LOCATE , 20
            PRINT “4 - Cancellazione”
            LOCATE , 20
            PRINT “5 - Recupero”
            LOCATE , 20
            PRINT “6 - Lista”
            LOCATE , 20
            PRINT “7 - Numero”
            LOCATE , 20
            PRINT “8 - Ricostruzione”
            LOCATE , 20
            PRINT “9 - Compattamento”
            PRINT
            LOCATE , 20
            PRINT “0 - Fine”
            Op$=INPUT$(1)
            SELECT CASE UCASE$(Op$)
                        CASE “1”
                                    AddPersona
                        CASE “2”
                                    Selec
                        CASE “3”
                                    Modify
                        CASE “4”
                                    Cancel
                        CASE “5”
                                    Recup
                        CASE “6”
                                    LstElenco
                        CASE “7”
                                    NumEls
                        CASE “8”
                                    ReIndex YesWait
                        CASE “9”
                                    Pack
                        CASE ELSE
END SELECT
BEEP
LOOP WHILE UCASE$(Op$)<>”0”
CLS
END
SUB AddPersona
DO
                        CLS
                        PRINT “(Inserimento)”
                        PRINT
                        InpDat
```

```
                Persona.Deleted = " "
                Nelements= Nelements+1
                Nf=FREEFILE
                OPEN "R", Nf, "AGE.DAT", LEN(Persona)
                PUT #Nf, Nelements, Persona
                CLOSE Nf
                OldNel=Nelements
                REDIM Index(1 TO Nelements) AS Idx
                ReadIndex ReRead
                Nelements = OldNel
                Index(Nelements).Cognome = Persona.Cognome
                Index(Nelements).Nome = Persona.Nome
                Index(Nelements).Record = Nelements
                WriteIndex
                PRINT
                INPUT "Un'altra immissione ? ", Altra$
                Altra$=LEFT$(UCASE$(Altra$), 1)
LOOP WHILE Altra$="S"
PRINT
PRINT "(Ordinamento in corso)"
QSortIndex 1, Nelements
WriteIndex
END SUB
SUB Cancel
        Nf=FREEFILE
        OPEN "R", Nf, "AGE.DAT", LEN(Persona)
        CLS
        PRINT "(Cancellazione)"
        PRINT
        PRINT "Precedente contenuto del record"
        PRINT
        VisNom CurrEl, Nf
        IF Persona.Deleted = " " THEN
                DO
                        INPUT ; "Si vuole cancellare il record ? ", VCan$
                        VCan$=LEFT$(UCASE$(VCan$), 1)
                LOOP WHILE INSTR("SN", VCan$) = 0
                IF VCan$="S" THEN
                        Persona.Deleted = "*"
                        PUT #Nf, Index(CurrEl).Record, Persona
                        LOCATE , 1
                        PRINT "Cancellazione effettuata" + SPACE$(50)
                END IF
        ELSE
                A$=INPUT$(1)
        END IF
        CLOSE Nf
END SUB
```

```
SUB InpDat
	LINE INPUT ; "Cognome   : "; Persona.Cognome
	VisDat Persona.Cognome
	LINE INPUT ; "Nome        : ";Persona.Nome
	VisDat Persona.Nome
	LINE INPUT ; "Indirizzo   : "; Persona.Indirizzo
LINE INPUT ; "Localita'   : "; Persona.Localita
	LINE INPUT ; "Provincia   : "; Persona.Provincia
LINE INPUT ; "CAP          : "; Persona.Cap
	LINE INPUT ; "Tel. Ab.   : "; Persona.TelefonoCasa
	LINE INPUT ; "Tel. Lav.   : "; Persona.TelefonoLavoro
END SUB
SUB LstElenco
	Nf=FREEFILE
	OPEN "R", Nf, "AGE.DAT", LEN(Persona)
	IF Nelements > 0 THEN
		FOR Rec=1 TO Nelements
			CLS
			PRINT "(Elenco)"
			PRINT
			LOCATE 3
			VisNom Rec, Nf
			A$=INPUT$(1)
			IF A$=CHR$(27) THEN
				EXIT FOR
			END IF
		NEXT Rec
	ELSE
		PRINT
		PRINT "Non esiste nessun nominativo."
		A$=INPUT$(1)
	END IF
	CLOSE #Nf
END SUB
SUB Modify
	Nf=FREEFILE
	OPEN "R", Nf, "AGE.DAT", LEN(Persona)
	CLS
	PRINT "(Modifica)"
	PRINT
	PRINT "Precedente contenuto del record"
	PRINT
	VisNom CurrEl, Nf
	IF Persona.Deleted = " " THEN
		DO
			INPUT ; "Si vogliono apportare modifiche ? ", VMod$
			VMod$=LEFT$(UCASE$(VMod$), 1)
		LOOP WHILE INSTR("SN", VMod$) = 0
```

```
			IF VMod$="S" THEN
				LOCATE , 1
				PRINT "Nuovo contenuto del record" + SPACE$(50)
				PRINT
				InpDat
				PUT #Nf, Index(CurrEl).Record, Persona
				Index(CurrEl).Cognome = Persona.Cognome
			Index(CurrEl).Nome = Persona.Nome
			END IF
			QSortIndex 1, Nelements
			WriteIndex
			CLOSE Nf
		ELSE
			A$=INPUT$(1)
		END IF
END SUB
SUB NumEls
		PRINT
		IF Nelements = 0 THEN
			PRINT "Non esiste nessun nominativo."
		ELSE
			Vb$="Esiste"
			Sp$="o."
			IF Nelements>1 THEN
				Vb$="Esistono"
				Sp$="i."
			END IF
			PRINT Vb$; Nelements; " nominativ" + Sp$
		END IF
		A$=INPUT$(1)
END SUB
SUB Pack
		CLS
		PRINT
		PRINT "(Compattamento file dati)"
		Nf1 = FREEFILE
		OPEN "R", Nf1, "AGE.DAT", LEN(Persona)
		Nf2 = FREEFILE
		OPEN "R", Nf2, "AGE.$$$", LEN(Persona)
		Nels=LOF(NF1)/LEN(Persona)
		IF Nels>0 THEN
			RecTo=0
			FOR RecFrom=1 TO Nels
				LOCATE 5
				PRINT "Record n. "; RecFrom
				GET #Nf1, RecFrom, Persona
				IF Persona.Deleted=" " THEN
					RecTo= RecTo+1
```

```
                        PUT #Nf2, RecTo, Persona
                    END IF
                NEXT RecFrom
                CLOSE #Nf1, #Nf2
                KILL "AGE.DAT"
                NAME "AGE.$$$" AS "AGE.DAT"
                ReIndex NoWait
ENDIF
        CLOSE #Nf1, #Nf2
        PRINT "(Compattamento effettuato)"
        A$=INPUT$(1)
END SUB
SUB QSortIndex(Sinistra, Destra)
        DIM Elx AS STRING * 51
        I=Sinistra
        J=Destra
        M=(Sinistra+Destra)/2
        Elx=Index(M).Cognome + " " + Index(M).Nome
        DO
                WHILE Index(I).Cognome + " " + Index(I).Nome < Elx AND I<Destra
                        I = I + 1
                    WEND
                WHILE Index(J).Cognome + " " + Index(J).Nome > Elx AND J>Sinistra
                        J = J  - 1
                    WEND
                    IF I <= J THEN
                        SWAP Index(I), Index(J)
                        I = I + 1
                        J = J - 1
                    END IF
        LOOP WHILE I <= J
        IF Sinistra < J THEN
                    QSortIndex Sinistra, J
        END IF
        IF Destra > I THEN
                    QSortIndex I, Destra
        END IF
END SUB
SUB ReadIndex (Rtyp%)
        Nf = FREEFILE
        OPEN "B", #Nf, "AGE.IDX"
        Nelements=LOF(Nf)\LEN(Index(1))
        CLOSE #Nf
        IF Nelements>0 THEN
                IF Rtyp=ReadFirst THEN
                    REDIM Index(1 TO Nelements) AS Idx
                END IF
                Segment=VARSEG(Index(1))
```

```
			Offset=VARPTR(Index(1))
			DEF SEG = Segment
			BLOAD "AGE.IDX", Offset
			DEF SEG
		END IF
END SUB
SUB Recup
		Nf=FREEFILE
		OPEN "R", #Nf, "AGE.DAT", LEN(Persona)
		CLS
		PRINT "(Recupero)"
		PRINT
		PRINT "Precedente contenuto del record"
		PRINT
		VisNom CurrEl, Nf
		IF Persona.Deleted = " " THEN
			PRINT "(Record non cancellato)"
		ELSE
			DO
				INPUT ; "Si vuole recuperare il record ? ", VRec$
				VMod$=LEFT$(UCASE$(VRec$), 1)
			LOOP WHILE INSTR("SN", VRec$) = 0
			IF VRec$="S" THEN
				Persona.Deleted = " "
				PUT #Nf, Index(CurrEl).Record, Persona
				LOCATE , 1
				PRINT "Recupero effettuato" + SPACE$(50)
			END IF
		END IF
		A$=INPUT$(1)
		CLOSE #Nf
END SUB
SUB ReIndex(FWait%)
		IF FWait THEN
			CLS
		END IF
		PRINT
		PRINT "(Ricostruzione indice)"
		Nf=FREEFILE
		OPEN "B", #Nf, "AGE.IDX"
		CLOSE #Nf
		KILL "AGE.IDX"
		Nf=FREEFILE
		OPEN "R", #Nf, "AGE.DAT", LEN(Persona)
		Nelements=LOF(Nf)/LEN(Persona)
		IF Nelements>0 THEN
			REDIM Index(1 TO Nelements) AS Idx
			FOR Rec=1 TO Nelements
```

```
                LOCATE 5
                PRINT "Record n. "; Rec
                GET #Nf, Rec, Persona
                Index(Rec).Cognome=Persona.Cognome
                Index(Rec).Nome=Persona.Nome
            Index(Rec).Record=Rec
            NEXT Rec
            PRINT
            PRINT "(Ordinamento in corso)";
            LOCATE , 1
            QSortIndex 1, Nelements
            WriteIndex
        END IF
        CLOSE #Nf
        PRINT "(Ricostruzione effettuata)"
        PRINT
        IF FWait THEN
            A$=INPUT$(1)
        END IF
  END SUB
SUB Selec
  DIM SCognome AS STRING * 25
  DIM SNome AS STRING * 25
  Nf=FREEFILE
  OPEN "R", #Nf, "AGE.DAT", LEN(Persona)
  DO
        CLS
        PRINT "(Selezione)"
        PRINT
        LINE INPUT ; "Cognome   : "; SCognome
        IF LEN(RTRIM$(SCognome)) = 0 THEN
            EXIT DO
        END IF
        VisDat SCognome
        LINE INPUT ; "Nome      : "; SNome
        VisDat SNome
        MinEl = 1
        MaxEl = Nelements
        Found = Falso
        IF Nelements>0 THEN
            DO
                CurrEl= ((MaxEl - MinEl) \ 2) + MinEl
                Cequ=RTRIM$(SCognome)=RTRIM$(Index(CurrEl).Cognome)
                Nequ=Vero
                IF LEN(RTRIM$(SNome))>0 THEN
                    Nequ=RTRIM$(SNome)=RTRIM$(Index(CurrEl).Nome)
                END IF
                IF Cequ AND Nequ THEN
```

```
                    Found = Vero
               ELSE
                    Dat1$=RTRIM$(SCognome) + " " + RTRIM$(SNome)
                    Dat2$=RTRIM$(Index(CurrEl).Cognome) + " "
                    Dat2$=Dat2$+RTRIM$(Index(CurrEl).Nome)
                    IF Dat1$>Dat2$ THEN
                         MinEl=CurrEl + 1
                    ELSE
                       MaxEl=CurrEl - 1
                    END IF
               END IF
          LOOP WHILE MaxEl-MinEl >= 0 AND NOT Found
END IF
PRINT
IF Found THEN
          PRINT "(Nominativo trovato)"
          PRINT
          VisNom CurrEl, Nf
ELSE
          PRINT "(Nominativo non trovato)"
END IF
Ex = Falso
Menu = Falso
DO
          PRINT
          PRINT "(P)recedente (S)ucessivo (R)icerca (M)enu"
          DO
               A$=UCASE$(INPUT$(1))
          LOOP WHILE INSTR("PSRM", A$) = 0
          SELECT CASE A$
               CASE "P"
                    IF CurrEl=1 THEN
                         BEEP
                         PRINT
                         PRINT "(Primo nominativo)"
                    ELSE
                         PRINT
                         CurrEl=CurrEl-1
                         VisNom CurrEl, Nf
                    END IF
               CASE "S"
                    IF CurrEl = Nelements THEN
                         BEEP
                         PRINT
                         PRINT "(Ultimo nominativo)"
                    ELSE
                         PRINT
                         CurrEl= CurrEl+1
```

```
                                        VisNom CurrEl, Nf
                                END IF
                        CASE "M"
                                Ex=Vero
                                Menu=Vero
                                EXIT DO
                        CASE "R"
                                Ex=Vero
                                EXIT DO
                END SELECT
        LOOP WHILE NOT Ex
        IF Menu THEN
                EXIT DO
        END IF
  LOOP
  CLOSE #Nf
END SUB
  SUB VisDat(Stri$)
        Stri$=UCASE$(LEFT$(Stri$, 1)) + LCASE$(MID$(Stri$, 2))
        COLOR 15
        LOCATE , 13
        PRINT Stri$
        COLOR 7
  END SUB
  SUB VisNom(Neleme%, Nfile%)
        GET #Nfile, Index(Nelem).Record, Persona
        IF Persona.Deleted = "*" THEN
                PRINT "(Record cancellato)"
                PRINT
                COLOR 15
                PRINT "("; Index(Nelem).Cognome; ")"
                PRINT "("; Index(Nelem).Nome; ")"
                COLOR 7
                PRINT
        ELSE
                PRINT "Cognome    : "; Persona.Cognome
                PRINT "Nome       : "; Persona.Nome
                PRINT "Indirizzo  : "; Persona.Indirizzo
                PRINT "Localita'  : "; Persona.Localita
                PRINT "Provincia  : "; Persona.Provincia
                PRINT "CAP        : "; Persona.Cap
                PRINT "Tel. Ab.   : "; Persona.TelefonoCasa
                PRINT "Tel. Lav.  : "; Persona.TelefonoLavoro
                PRINT
        END IF
  END SUB
  SUB WriteIndex
        IF Nelements > 0 THEN
```

```
                Segment = VARSEG(Index(1))
                Offset = VARPTR(Index(1))
                Length=Nelements * LEN(Index(1))
                DEF SEG=Segment
                BSAVE "AGE.IDX", Offset, Length
                DEF SEG
        END IF
    END SUB
```

6.4.2 Condivisione di file in multiutenza

Quando si lavora in un ambiente in cui i dati registrati su disco possono essere condivisi da diversi utenti, nascono dei nuovi, particolari problemi nella loro gestione, problemi che, in monoutenza, non erano affatto considerati.

Sia che si lavori con una **rete locale** che con un terminale collegato ad un elaboratore centrale che usi usi un sistema operativo multiutente, quale **XENIX** o **UNIX**, può accadere di dover leggere o scrivere in un file comune a diversi utenti.

È classico l'esempio dei terminali posti in ogni agenzia di banca che lavorano con dei file comuni posti all'interno dei dischi dell'elaboratore centrale dell'istituto, molte volte, posto a parecchi chilometri da questi. O anche i terminali delle agenzie di viaggio, collegati con i sistemi di elaborazione delle compagnie aeree, che sfruttano, tutti, gli stessi dati relativi alle prenotazionei posti in archivi centrali.

Il problema fondamentale di questo tipo di sistemi deriva dal fatto che, in un determinato istante, **più terminali** possano volere accedere, in scrittura, ad uno stesso gruppo di dati. Se un determinato gruppo di dati sta per essere modificato, non è possibile che un altro utente li richieda, in quel momento, per modificarli a sua volta ma, al contrario, quest'ultimo deve essere costretto ad aspettare che i dati vengano resi nuovamente disponibili. Ad esempio, se un utente A sta modificando dei dati relativi alla prenotazione di 2 posti sul volo AZxxx odierno, **blocca** l'utilizzo del file per permetterne l'aggiornamento avendo la garanzia che un utente B, trovando lo stesso file bloccato, non possa modificarlo finché il primo utente non termina l'operazione.

A quel punto, l'utente A **sblocca** il file e permette ad un altro qualsiasi utente di usarlo. Tale blocco può essere fatto sia a livello di file che a livello di record; è cioè possibile non rendere disponibile a tutti gli utenti un solo determinato record di un file su cui si sta lavorando in quel momento.

Generalmente le operazioni di blocco sono, nel tempo, molto limitate, tanto da permettere l'aggiornamento sicuro dei dati; questo evita che un utente impieghi molto tempo per sfruttare un file dati.

In Quick Basic sono presenti due istruzioni, **LOCK** ed **UNLOCK** (v. 5.1.1), tramite le quali, dopo averlo aperto è possibile bloccare e sbloccare un deteriminato file o un record di tale file. Per il loro utilizzo si rimanda alla descrizione nel capitolo 5.

Tenere presente che, in qualche sistema operativo multiutente, pur essendo obbligatorio il bloccaggio di un record, esso viene automaticamente sbloccato dallo stesso S.O. subito dopo l'aggiornamento su disco. È comunque buona norma prevedere sempre l'istruzione di sbloccaggio per rendere il codice più portabile.

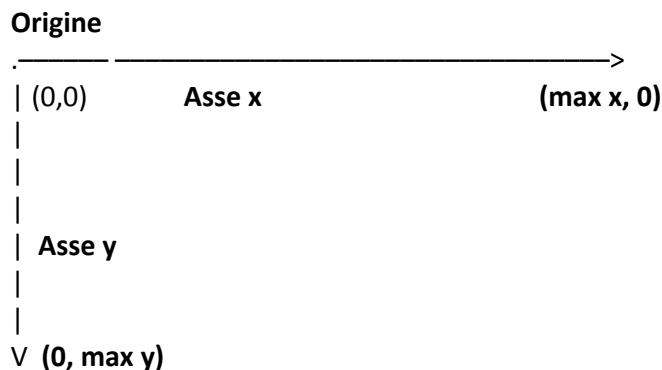
6.5 LA GRAFICA

6.5.1 Cenni sulle potenzialità grafiche

Il Quick Basic dispone di istruzioni e funzioni adatte alla gestione della grafica, quali

SCREEN, PSET, PRESET, LINE
CIRCLE, DRAW, VIEW, WINDOW
PMAP, POINT (), COLOR, PALETTE
PAINT, GET, PUT, PCOPY

Per potere utilizzare tali istruzioni, tuttavia, è necessario che il computer abbia montata una scheda video che permetta la visualizzazione grafica, in uno degli standard attualmente più usati (CGA, EGA, VGA, Hercules); in caso contrario, il semplice tentativo di utilizzo di tali capacità, causa l'emissione di un messaggio di errore.
Tramite l'istruzione SCREEN, seguita da un valore numerico, è possibile scegliere il tipo di schermo grafico su cui visualizzare i propri dati, ed in base a tale tipo, si possono sfruttare più o meno punti sullo schermo e più o meno colori per ogni punto. Ad esempio, nel modo 1, per computer dotati, almeno, di scheda grafica CGA, è possibile trattare 320 punti (**pixels**) in orizzontale e 200 punti in verticale in 4 colori, mentre, con la modalità 2 e la stessa scheda, si raggiungono i 640 punti in orizzontale per 200 in verticale, ma con due soli colori.

Con altre schede ed altri modi, è possibile raggiungere capacità di rappresentazione grafica molto elevate (scheda VGA e SVGA) e con molti colori, ma l'utilizzo delle principali istruzioni grafiche del Quick Basic resta immutato.
Una volta attivata la visualizzazione grafica, lo schermo del sistema diventa come un piano delimitato da assi cartesiani la cui origine però, è posta in alto a sinistra

La coordinata x dei punti sull'ultima colonna a destra, quindi, è determinata dal numero massimo di pixels che è possibile visualizzare, in senso orizzontale, con la modalità prescelta, e lo stesso discorso vale per la coordinata y dei punti posti sull'ultima riga, rispettivamente, con il massimo numero di punti che è possibile visualizzare in verticale.

Tutti i punti intermedi, quindi, hanno coordinate variabili da 0 ai massimi suddetti e tali coordinate vanno espresse nelle istruzioni e funzioni che lo richiedono quando si deve trattare un singolo punto. Ad esempio, l'istruzione **PSET** prevede che vengano precisate tali coordinate per permettere che il punto corrispondente venga colorato con il colore di primo piano corrente. Lo stesso per l'istruzione **PRESET** tranne per il fatto che il colore adottato, in questo caso, è quello di sfondo; in questa maniera l'istruzione PRESET permette di cancellare i punti colorati con l'istruzione PSET e questo fatto è utile negli algoritmi di animazione.
Tuttavia, con le istruzioni suddette, è possibile indicare anche un colore diverso da quelli usati per default, per colorare un punto determinato dalle sue coordinate.

L'istruzione **LINE** è invece usata per tracciare un segmento, dati i due estremi, nel colore e con le modalità prescelte dall'utente (v. istr. LINE 5.1.1).
Nell'esempio seguente è mostrato un programma che usa tali istruzioni, insieme a molte altre, per visualizzare il grafico di una funzione

```
DEF FNUser(X) = SIN(X)
DO
        SCREEN 0
        CLS
        INPUT "Estremo sinistro    : ", Esx
        INPUT "Estremo destro      : ", Edx
        INPUT "Estremo inferiore   : ", Ein
        INPUT "Estremo superiore : ", Esu
        SCREEN 2
        WINDOW (Esx, Ein)-(Edx, Esu)
        FOR Cx=Esx TO Edx STEP (Edx-Esx) / 500
                PSET (Cx, FNUser(Cx))
        NEXT Cx
LOOP WHILE INKEY$<>""
A$=INPUT$(1)
SCREEN 0
END
```

Per quanto riguarda l'uso delle altre istruzioni (e funzioni) relative alla gestione della grafica, si rimanda alla loro descrizione fornita nel capitolo 5.

CAPITOLO 7

Il QB e gli altri linguaggi

CAPITOLO 7

7.1 QBASIC, C ED ASSEMBLER

7.1.1 Generalità sull'interfacciamento ad altri linguaggi

Durante la realizzazione di un programma, con qualsiasi linguaggio, può rendersi necessario l'uso dell'Assembler per la realizzazione di particolari parti dello stesso che devono essere molto veloci ed efficienti. Lo stesso discorso vale per il linguaggio C, di cui molti programmatori conosceranno le potenzialità, tramite le cui librerie è possibile realizzare programmi molto complessi in maniera rapida.

Il QB permette la realizzazione di un programma con linguaggi misti e l'esecuzione dello stesso usando sia il compilatore BC da DOS, sia l'ambiente e le Quick-Libraries. Da DOS, le routines realizzate in altri linguaggi possono essere riunite in un unico file di libreria e per questo la Microsoft mette a disposizione il programma LIB (v 4.2).

Porre attenzione al fatto che, tutte le routines incluse in una Quick Library, devono essere controllate **prima** del loro uso con un programma, dato che il debug al loro interno non è possibile.

Per tali programmi con linguaggi misti, bisogna considerare tre punti fondamentali per la loro corretta realizzazione, e cioè

- convenzioni per la formazione dei nomi delle procedure
- convenzioni per la chiamata delle procedure
- convenzioni per il passaggio dei parametri tra procedure

In generale, infatti, la realizzazione di programmi con linguaggi misti prevedono la chiamata **da Quick Basic** di funzioni o procedure scritte in altri linguaggi ed, eventualmente, un preventivo passaggio di dati necessari all'elaborazione.

Naturalmente, usando altri linguaggi, è necessario avere a disposizione anche i compilatori relativi affinché possa essere creato un file OBJ (file oggetto) dal file sorgente corrispondente (ASM o C). Tale file oggetto, se derivante dal Microsoft C Compiler, dal Microsoft Quick C Compiler o dal Microsoft MacroAssembler, è compatibile con quelli generati dal Microsoft Quick Basic e, quindi, tranquillamente utilizzabile.
Non è così, almeno in generale, per altri compilatori, che, anche se non sempre, generano dei file oggetto non completamente compatibili, che per essere utilizzati, devono essere modificati.
L'unione dei file oggetto, comunque, è possibile utilizzando il programma **LINK** (v. 4.1.2) della Microsoft; questo provvede ad organizzare tutti i file oggetto ed eventuali librerie, in un file eseguibile (EXE) da MS-DOS.

Le routines che sono richiamate tra i vari linguaggi, possono ritornare un valore; in questo caso bisogna fare delle precisazioni sul tipo di routine che bisogna dichiarare a seconda del tipo di

linguaggio usato. La seguente tabella mostra, per ogni linguaggio, il tipo di routine da utilizzare a seconda del loro valore di ritorno della stessa

	BASIC	C	MacroAssembler
Valore ritornato	FUNCTION	function	Proc
Nessun valore ritornato	SUB	(void)function	Proc

Quindi, ad esempio, per richiamare da QB una routine scritta in C che ritorni un valore, questa deve essere dichiarata, nel file sorgente C, come funzione con la parola chiave che determini il tipo di dato ritornato; nal caso in cui non debba essere ritornato alcun valore, questa parola chiave deve essere **void**. Per il MacroAssembler, invece, non esiste differenza, dato che le Proc (procedure assembler) ritornano comunque un valore; esso viene semplicemente ignorato dal QB, quando è il caso.

- Convenzioni per la formazione dei nomi delle procedure

È importante precisare il nome corretto di tutte le routines chiamate all'interno di una procedura scritta in linguaggio misto; in caso contrario il programma LINK, all'atto dell'unione di tutti i file oggetto, non trovando in nessuno di questi una routine richiamata in un altro punto, emette un messaggio di errore (Unresolved external) e non consente di creare un file eseguibile corretto. L'errore suddetto è determinato dal fatto che, per il LINK, la routine il cui nome compare in una chiamata fatta in un qualsiasi file oggetto, viene considerata **esterna** (presente in altri file non specificati), e quindi tale situazione viene considerata non risolvibile.

Il QB adotta le seguenti convenzioni per la formazione di un nome di routine

- la lunghezza massima consentita è di 40 caratteri
- tutti i caratteri sono convertiti in maiuscolo
- viene eliminata il carattere di tipo (%,&,!,#,$)

Per il linguaggio C, invece, le convenzioni sono diverse, e cioè

- la lunghezza massima del nome è di 31 caratteri
- i caratteri sono minuscoli
- viene posto il carattere _ (underscore) davanti al nome

Per il MacroAssembler, invece, non esistono particolari problemi fatta eccezione per la loro lunghezza massima, che è di 31 caratteri.

7.1.2 Interfacciamento al MacroAssembler v. 5.0. Microsoft

L'interfacciamento tra i due linguaggi è possibile unicamente quando, in un programma scritto in Quick Basic, si faccia riferimento a procedure appartenenti ad un testo scritto e compilato con il MacroAssembler. Nel caso inverso, quando cioè si vuole richiamare delle procedure

scritte in Quick Basic da programmi Assembler, la situazione presenta tali difficoltà da rendere la cosa impossibile.
Del resto la maggior parte delle volte l'interfaccia è del primo tipo; così, infatti, si può ottenere

- una parte di programma che venga eseguito ad una velocità molto elevata;

- l'accesso a risorse di sistema in altro modo non utilizzabili.

In un testo Assembler che rispetti tutte le regole per un corretto interfacciamento dei linguaggi, sono necessarie diverse parti, e cioè

- Inizializzazione delle procedure
- Immissione della procedura
- Allocazione dei dati locali
- Conservazione dei registri di lavoro
- Accesso ai parametri
- Emissione di un valore di ritorno
- Uscita dalla procedura

Ognuna delle fasi suddette deve essere quindi esaminate nei dettagli.

- **Inizializzazione delle procedure**

Il MacroAssembler v. 5.0. della Microsoft permette la definizione del modello prescelto ed i segmenti in maniera automatica e compatibile con il Quick Basic. Infatti, il Quick Basic prevede il modello **medio** di indirizzamento delle routines e dei dati (routines di tipo FAR e dati di tipo NEAR), e ciò si ottiene tramite la pseudo istruzione .MODEL del seguente modo

.MODEL MEDIUM

In tale modo tutte le procedure saranno, automaticamente, dichiarate di tipo FAR e ciò permetterà la corretta interfaccia tra i due linguaggi. Ovviamente, se si fosse in possesso di una precedente versione di MacroAssembler, si dovrebbero esplicitamente dichiarare tutte le procedure richiamate dal QB di tipo FAR.

Per quanto riguarda la dichiarazione dei segmenti da utilizzare nel programma assembler, con il MASM 5.0., è sufficiente la pseudo istruzione .CODE, per il testo, e .DATA per i dati eventuali. La seguente tabella riepiloga tutte le pseudo istruzioni che è possibile usare ed il relativo effetto

Directive	Name	Align	Combine	Class	Group
.CODE	name_TEXT	WORD	PUBLIC	'CODE'	——
.DATA	_DATA	WORD	PUBLIC	'DATA'	DGROUP
.CONST	CONST	WORD	PUBLIC	'CONST'	DGROUP
.DATA?	BSS	WORD	PUBLIC	'BSS'	DGROUP
.STACK	STACK	PARA	STACK	'STACK'	DGROUP

I segmenti definiti tramite le precedenti pseudoistruzioni sono i seguenti

* .CODE : Segmento contenente tutto il codice
* .DATA : Segmento contenente i dati inizializzati
* .DATA? : Segmento contenente i dati non inizializzati
* .CONST : Segmento contenente tutte le costanti
* .STACK : Segmento di stack

Tutti i segmenti contenenti dati, compreso lo stack, sono opzionali e, se dichiarati, vengono associati al gruppo DGROUP.

Nella versione precedente di MacroAssembler, tale associazione deve essere esplicitamente effettuata (pseudoistruzione GROUP), così come deve essere usata la frase ASSUME per garantire l'indirizzabilità dei dati.

Per terminare la fase di inizializzazione del programma è necessario dichiarare pubbliche tutte le procedure richiamate da QB perché il LINK possa effettuare correttamente i collegamenti; ciò è possibile tramite la pseudoistruzione PUBLIC.
Inoltre, tutti i riferimenti a routines o dati globali esterni al programma assembler devono essere dichiarati con la pseudoistruzione .EXTRN.

Lo schema essenziale della parte relativa all'inizializzazione di un testo di un programma assembler da collegare al QB, che non usi dati propri è quindi il seguente

```
.MODEL MEDIUM
.CODE
PUBLIC nomeproc
```

in cui 'nomeproc' è il nome della procedura che deve essere inserita in seguito.

- **Immissione della procedura**

Dopo l'inizializzazione, nel testo Assembler va inserita la procedura che deve realizzare quanto richiesto dal programmatore. Ad essa va dato il nome specificato dopo la pseudoistruzione PUBLIC e va scritta tra la PROC e la ENDP, come nell'esempio seguente

```
            .MODEL MEDIUM
            .CODE
            PUBLIC nomeproc
nomeproc    PROC
            …
nomeproc    ENDP
```

eventuali procedure richiamate, a loro volta, dalla procedura principale, non devono essere, necessariamente, dichiarate pubbliche e possono essere poste dopo la fine della procedura principale e prima della fine del testo (dopo la ENDP della procedura principale e prima della END di chiusura).

- **Allocazione dei dati locali**

Se necessario, è possibile preparare dello spazio per dati da usarsi all'interno delle procedure Assembler. Questo spazio può essere direttamente allocato al momento della compilazione del testo (dati di tipo statico) e posto all'interno di un segmento definito con la pseudoistruzione .DATA. Certe volte si preferisce invece, allocare lo spazio al momento dell'esecuzione delle procedure e liberare lo stesso alla loro fine (dati di tipo automatico); in questo caso viene impegnato lo **stack** e ciò si ottiene nel seguente modo

```
push bp           Conserva il registro bp
mov bp,sp         Copia sp in bp per utilizzarlo come base
sub sp,nbytes     Preserva n bytes nello stack
    …
mov sp,bp
    …
```

In questa maniera, alla chiamata della procedura Assembler, il contenuto dello stack è quindi il seguente

Indirizzi alti

Parametri	**(2 bytes * n parametri)**
Indirizzo di ritorno al QB	**(4 bytes)**
Registro BP	**(2 bytes)**
Area dati locali	**(n bytes)**
Copie dei registri	**(2 bytes * n registri)**

Indirizzi bassi

L'area allocata per i dati locali viene indirizzata tramite il registro puntatore BP; la prima coppia di bytes è indirizzata da BP-2, la seconda da BP-4 e così via. I parametri passati dal QB, invece, sono indirizzati da BP+6, BP+8 e così via.

Lo spazio allocato al momento della compilazione viene inserito, secondo le regole dettate dal Macroassembler, nel segmento dei dati associato alla pseudoistruzione .DATA, come nel seguente esempio

```
.MODEL MEDIUM
.DATA
DatoA  DW 0
DatoB  DW 0
.CODE
PUBLIC nomeproc
nomeproc PROC
    …
nomeproc ENDP
END
```

In questo caso è conveniente indirizzare tali dati servendosi del registro DS ed i dati che sono contenuti nel segmento del QB, con il registro ES. A tale fine, si deve copiare in ES, all'inizio

della procedura, il registro DS e caricare in quest'ultimo il valore del segmento dei dati. Prima di tutto ciò è conveniente salvare in due zone di memoria appositamente create, il contenuto precedente dei due registri. Il seguente pezzo di codice mostra quanto detto

```
        .MODEL MEDIUM
        .DATA
        OLDDS DW ?
        OLDES DW ?

                ...     (Altri dati da potere utilizzare)

        .CODE
        PUBLIC Prova
Prova   PROC

        ; Salva e prepara i registri di segmento

        mov bx,ds
        mov ax,@DATA
        mov ds,ax
        mov OLDDS,bx
        mov OLDES,es
        mov es,bx

                ...     (Altro codice per la procedura)

        ; Riprende i vecchi registri di segmento

        mov es,OLDES
        mov ds,OLDDS
        ret

Prova   ENDP
        END
```

I parametri, del resto, possono essere anche prelevati tramite le istruzioni 'pop' e registrati nell'area dati inizializzata in precedenza con .DATA. Tenere presente che, normalmente, i dati sono passati per **indirizzi**; viene cioè passato dal QB l'indirizzo (offset) relativo al segmento DS del parametro richiesto nell'ordine opposto all'inserimento dei parametri specificati nell'istruzione di chiamata scritta in QB. Per questo motivo è necessario usare un altro registro (normalmente bx) come indice per l'estrazione dalla memoria del valore effettivo dei parametri. Tali valori possono essere così memorizzati nell'area dei dati della procedura. Se si sono trattati i registri dei segmenti come detto in precedenza, tali indirizzi saranno riferiti al registro ES ed il seguente esempio mostra come è possibile, con questo metodo, passare gli argomenti dal QB

```
                pop ax
                pop dx
```

```
pop bx
mov bx,es:[bx]
mov Par2,bx
pop bx
mov bx,es:[bx]
mov Par1,bx
       …
push dx
push ax
```

In questo caso bisogna precisare che i valori dei parametri sono depositati in zone di memoria di 2 bytes ciascuna definite nel segmento dei dati (pseudoistruzione .DATA); inoltre, vengono prelevati dallo stack due bytes (in DX e in AX) temporaneamente per poi essere reinseriti nello stack dopo l'estrazione dei parametri; tali bytes rappresentano l'indirizzo di rientro (segmento:offset) al programma in QB ed essendo posti prima dei parametri, se trattati diversamente, potrebbero creare dei problemi.
Operando in tale modo si evita di dovere porre alla fine della procedura l'istruzione **ret nbytes** ma basta la sola **ret**, dato che lo stack è liberato al momento del prelevamento dei parametri.
Solo se i parametri sono passati per **valore** allora è sufficiente, per ognuno di essi, il seguente codice

```
…
pop Par2
pop Par1
…
```

che pone il valore direttamente nella zona di memoria riservata ai dati per la procedura Assembler.

Nel programma scritto in Quick Basic, è necessaria la frase DECLARE per la dichiarazione della procedura Assembler; nella frase suddetta è possibile specificare anche tutti i parametri che dovranno essere passati tra i linguaggi. Questi saranno passati nello stack in maniera che l'ultimo parametro sarà il primo disponibile nello stack. Se nella frase DECLARE si dovesse usare l'opzione **CDECL** (v. Cap. 5), i parametri sarebbero passati secondo le convenzioni del linguaggio C e cioè in maniera inversa (il primo parametro della procedura è il primo disponibile nello stack). Se si dovesse usare tale opzione inoltre, si dovrebbe tenere conto dei seguenti fatti

- viene aggiunto un carattere _ (underscore) al nome della routine e per evitare problemi con il programma LINK diventa necessario usare l'opzione ALIAS (v. Cap. 5);

- non si deve terminare la procedura con l'istruzione **RET Size** ma solo con la **RET** dato che lo stack viene messo a posto automaticamente;

- è necessario usare tale opzione quando si devono passare degli argomenti il cui numero è variabile.

In generale l'uso dell'opzione CDECL non è molto frequente, ed i pochi casi in cui essa potrebbe essere usata, possono essere risolti anche in sua mancanza.

- **Conservazione dei registri di lavoro**

È buon norma, prima del codice della procedura, inserire delle istruzioni per salvare i registri con cui si lavorerà (SI, DI, BP, …) per poi poterli riprendere alla fine della procedura. Anche se con il Quick Basic v. 4.0. e 4.5. **non è necessario** salvare tali registri (mentre con il 3.0. lo era), è comunque positivo includere tali istruzioni per essere sicuri che, in qualsiasi situazione, non possano esistere dei problemi.
I registri vanno salvati nello stack, dopo avere eventualmente allocato uno spazio locale (dopo l'istruzione sub sp,nbytes), nel seguente modo

```
…
push si
push di
…
```

e ripresi, nell'ordine inverso (per le caratteristiche dello stack), con

```
…
pop di
pop si
…
```

7.1.3 Esempi di interfacciamento al MASM v. 5.0

Il seguente esempio (commentato nei suoi blocchi)

```
; Routine EXPEXT - Versione 1.0. (EXPEXT.ASM)
; (da linkare con il programma in Quick Basic)
; A. Giuliana - Monreale 1991
;
; Viene usato il modello medio
.MODEL MEDIUM
; Inizio del segmento di codice della routine
; La routine viene dichiarata pubblica
.CODE
PUBLIC  Expext
ExpExt          PROC
                ; Preparazione registro bp per
                ; l'accesso ai parametri
                        push bp
                        mov bp,sp
                ; Viene controllata l'esistenza
                ; del gestore di memoria espansa
                        mov ah,35h
                        mov al,67h
                        int 21h
                        mov ax,0
```

```
                cmp word ptr es:[0Ah],4d45h
                jnz EXP
        ; Viene ricavata la dimensione (in k)
        ; della memoria espansa
                mov ah,42h
                int 67h
                mov ax,bx
                mov cx,4
                shl dx,cl
        ; Il valore (in K) della memoria espansa
        ; viene passato nella variabile del QB
                mov bx,[bp+0Ah]
                mov [bx],dx
EXP:
                mov cx,4
                shl dx,cl
        ; Il valore (in K) della memoria espansa
        ; libera viene passato nella variabile del QB
                mov bx,[bp+8]
                mov [bx],ax
        ; Viene ricavata la dimensione (in k)
        ; della memoria estesa
                mov ah,88h
                int 15h
        ; Il valore (in K) della memoria estesa
        ; viene passato nella variabile del QB
                mov bx,[bp+6]
                mov [bx],ax
        ; Viene recuperato il registro bp
        ; e termina la routine con il
        ; rientro al Quick Basic
                pop bp
                ret 4
ExpExt  ENDP
```

mostra come è possibile scrivere una procedura collegata al QB in maniera che venga ricavata la grandezza della memoria espansa e della estesa del sistema su cui si lavora.
Dopo aver precisato il modello ed il segmento del codice (prime due pseudoistruzioni), viene dichiarata pubblica la procedura di nome **ExpExt** che inizia alla riga seguente.
Notare la mancanza di un segmento dati (pseudoistruzione .DATA) visto che la procedura non usa memoria per dei dati propri.
Le prime righe della procedura provvedono a salvare il registro bp e preparare il nuovo valore dello stesso per accedere ai parametri (ricordare che esiste un altro sistema per accedere ai parametri stessi), mentre le prossime istruzioni (fino alla prima int 21h) provvedono a ritornare nei registri ES:BX l'indirizzo della routine che provvede alla gestione della memoria **espansa**. Se nelle celle di memoria indirizzate da ES:000A ed ES:000B esiste la parola EM, allora il driver è attivato ed è possibile conoscere il valore della memoria espansa disponibile, altrimenti viene ritornato, nel registro AX il valore 0.

Se tale memoria è disponibile, invece, il valore ritornato viene moltiplicato per 16 (ogni pagina è fatta da 16K) ed il risultato, espresso in Kbytes, viene depositato agli indirizzi dei parametri passati dal QB in maniera da renderli disponibili al programma Basic. Notare che i registri di segmento non sono stati modificati perché non esistono dati locali alla procedura.
Per la memoria estesa, vale lo stesso discorso ad iniziare dall'istruzione **mov ah,88h**, ed il valore viene depositato all'indirizzo del primo parametro.

In Qb, un programma che sfrutti tale procedura, può essere, ad esempio, il seguente

```
' (Main Module PROMEM.BAS)
DEFINT A-Z
DECLARE SUB EXPEXT(TEXP, FEXP, FEXT)
CLS
EXPEXT TEXPMEM, FEXPMEM, FEXTMEM
PRINT "Memoria espansa totale (Kb) : "; TEXPMEM
PRINT "Memoria espansa libera (Kb) : "; FEXPMEM
PRINT "Memoria estesa libera (Kb)    : "; FEXTMEM
END
```

in cui, dopo essere state dichiarate tutte le variabili di tipo intero, viene specificato il nome della routine (EXPEXT) e che essa utilizza tre parametri (di tipo intero).
Dopo la cancellazione del video, viene richiamata tale routine e l'indirizzo di tre variabili viene passato alla EXPEXT; tali indirizzi saranno utilizzati per modificare il contenuto delle variabili stessi da parte del programma Assembler; nello stack i tre indirizzi saranno posti nel seguente ordine

bp+0ah	TEXPMEM
bp+08h	FEXPMEM
bp+06h	FEXTMEM

La compilazione di questo file viene fatto con il comando

BC PROMEM;

mentre la compilazione del testo assembler (file EXPEXT.ASM) con la linea

MASM EXPEXT;

I due file oggetto (PROMEM.OBJ e EXPEXT.OBJ) devono essere collegati, tramite il LINK, con la linea

LINK PROMEM EXPEXT;

Il file PROMEM.EXE risultante è correttamente eseguibile dal DOS.

In quest'altro esempio, un po' più complesso, viene realizzata una routine in assembler per il riempimento veloce di un array di interi a n dimensioni, con un qualsiasi numero compreso nel range degli interi. A questa routine, denominata ARRFILL, devono essere passati due parametri,

l’array da riempire ed il valore da depositare, ed a tale scopo, nel programma in Quick Basic, si usa la frase DECLARE con l’indicazione dell’array e del dato intero usato per riempirlo. Il fatto che entrambi o parametri siano interi viene assicurato, automaticamente, dalla frase DEFINT posta all’inizio; notare inoltre la presenza della pseudoistruzione DYNAMIC che consente di utilizzare arrays molto grandi (>64K) sfruttando tutta la memoria convenzionale a disposizione del sistema (infatti l’array usato come prova in questo esempio, occupa 200 x 200 x 2 = 80000 bytes).

```
‘ (Main Module PROARR.BAS)
‘ $DYNAMIC
DEFINT A-Z
DECLARE SUB ARRFILL(ARR(), MASK)
CLS
DIM X(1 TO 200, 1 TO 200)
INPUT “Valore : “, V
ARRFILL X(), V
FOR A1=1 TO 200
    FOR A2=1 TO 200
            PRINT X(A1, A2);
    NEXT A2
NEXT A1
END
```

La chiamata della routine assembler, a questo punto, è fatta, semplicemente, usando il nome della stessa (ARRFILL) come istruzione ed i parametri necessari (array X e variabile V); notare che, nei parametri, l’array è indicato dalle doppie parentesi e porre attenzione al fatto che il nome degli stessi può essere differente da quelli indicati nella frase DECLARE dato che, in tale dichiarazione ha importanza solo il tipo ed il numero di argomenti.
Il doppio circlo FOR che segue è usato solamente per controllare il contenuto dell’array ed evidenziare il funzionamento della routine. Per confronto, questi sono i tempi di esecuzione del programma che effettua l’operazione di riempimento dell’array usando la routine assembler o usando delle istruzioni in Quick Basic

Con la routine Assembler	**0,26 secondi**
In Quick Basic	**2,08 secondi**

È evidente quindi, come in questo caso, sia importante l’uso dei linguaggi misti in programmazione.
La routine in assembler, tutta ampiamente commentata, è la seguente

```
; Routine ARRFILL - Versione 1.1. (ARRFILL.ASM)
; (da linkare con il programma in Quick Basic)
; A. Giuliana - Monreale 1991
;
; Viene usato il modello medio
.MODEL MEDIUM
; Inizio del segmento dei dati della routine
.DATA
```

```
    OLDDS        DW ?
    OLDES        DW ?
    ARRSTR       DD ?
    ARRSTP       DD ?
    MSK          DW ?
; Inizio del segmento di codice della routine
; La routine viene dichiarata pubblica
.CODE
PUBLIC           ArrFill
ArrFill     PROC
    ; Preparazione registri di segmento
    ; DS ---> area dati nella routine
    ; ES ---> area dati del Quick Basic
            mov bx,ds
            mov ax,@DATA
            mov ds,ax
            mov OLDDS,bx
            mov OLDES,es
            mov es,bx
    ; estrazione parametri
            pop ax
            pop dx
    ; Viene estratto prima l'indirizzo
    ; e poi il valore della variabile
    ; usato come riempimento; tale valore
    ; viene posto nell'area assember MSK
            pop bx
            mov bx,es:[bx]
            mov MSK,bx
    ; Per l'array viene prelevato
    ; l'indirizzo della testata
    ; di descrizione dello stesso
    ; (tale indirizzo rimane in bx);
    ; viene posto l'indirizzo (seg:off)
    ; dell'array in ARRSTR
            pop bx
            push es
            les cx,es:[bx]
            mov word ptr ARRSTR,cx
            mov word ptr ARRSTR+2,es
            pop es
            push dx
            push ax
    ; Il registro cx viene caricato
    ; con il numero di dimensioni
    ; dell'array scritto in testata
            xor ch,ch
            mov cl,es:[bx+8]
```

```
    ; Il registro si viene usato per
    ; puntare nella testata al numeo
    ; degli elementi per ogni dimensione
            mov si,bx
            add si,0Eh
; Viene posto a 1 il totale iniziale
; degli elementi dell'array (32 bits)
            mov ax,1
            xor dx,dx
    ; Si effettuano tutte le moltiplicazioni
    ; per ricavare il numero degli elementi
NxtDim:
            mul word ptr es:[si]
            add si,4
            loop NxtDim
    ; Viene effettuato il calcolo dei bytes
    ; occupati dall'array (elementi x 2)
            shl sx,1
            rcl dx,1
    ; Nell'area ARRSTP viene posto l'indirizzo
    ; finale+1 dell'array (inizio in ARRSTR)
            add ax,word ptr ARRSTR
            adc dx,0
            mov word ptr ARRSTP,ax
            mov cl,0ch
            shl dx,cl
            adc dx,word ptr ARRSTR+2
            mov word ptr ARRSTP+2,dx
    ; Nel registro dx viene posto il valore
    ; usato come riempimento dell'array
            mov dx,MSK
    ; Il registro di viene usato come
    ; puntatore all'array da riempire
            mov di,word ptr ARRSTP
    ; Si decrementa il puntatore di 2 bytes
Minus2:
            push ds
            push word ptr ARRSTP+2
            pop ds
            sub di,2
            jnc NoPre
            pop ds
            sub word ptr ARRSTP+2,1000h
            mov ax,word ptr ARRSTP+2
            push ds
            mov ds,ax
    ; Viene depositato il valore di
    ; riempimento in un elemento dell'array
```

```
NoPre:
            mov [di],dx
    ; Il programma viene ripetuto solo
    ; se il puntatore attuale è diverso
    ; dall'indirizzo iniziale dell'array
            pop ds
            cmp di,word ptr ARRSTR
            jnz Minus2
            mov ax,word ptr ARRSTP+2
            cmp ax,word prt ARRSTR+2
            jnz Minus2
    ; Vengono recuperati i registri
    ; di segmento salvati all'inizio
    ; e termina la routine con il
    ; rientro al Quick Basic
NoEqu:
            mov es,OLDES
            mov ds,OLDDS
            ret
ArrFill     ENDP
    END
```

Notare che, per accedere ai parametri, in questo esempio, non si è fatto uso del registro bp, ma si è preferito usare l'istruzione pop che preleva dallo stack l'indirizzo necessario; in questo modo, inoltre, non è stato necessario specificare alla fine della routine il numero dei bytes da scartare nello stack con l'istruzione ret.
La procedura da seguire per l'unione dei due programmi, è la seguente

- compilazione del programma PROARR.BAS con la riga

BC /AH PROARR;

- compilazione del programma ARRFILL.ASM con la riga

MASM ARRFILL;

- unione dei due file OBJ generati in precedenza

LINK PROARR ARRFILL;

Il file PROARR.EXE così generato può essere eseguito sotto MS-DOS.

Il terzo esempio di interfacciamento del QB al MacroAssembler, riguarda una routine (MATADD) realizzata per effettuare la somma tra due matrici quadrate di ordine n. Gli indirizzi delle testate dei due arrays da sommare e di quello che deve contenere i risultati sono forniti alla routine, che esegue la somma richiesta. Il testo del programma scritto in QB che serve come prova della routine, è il seguente

```
‘ (Main Module PROMAT.BAS)
DEFINT A-Z
RANDOMIZE TIMER
‘ Dichiarazione della routine Assembler e dei parametri
DECLARE SUB MATADD (MATA(), MATB(), MATR())
CLS
‘ Dimensionamento degli arrays (M3() <--- M1() + M2())
DIM M1(1 TO 10, 1 TO 10)
DIM M2(1 TO 10, 1 TO 10)
DIM M3(1 TO 10, 1 TO 10)
‘ Caricamento delle matrici da sommare
‘ con valori scelti a caso
FOR X=1 TO 10
        FOR Y=1 TO 10
                M1(X, Y)=RND*50
                M2(X, Y)=RND*50
        NEXT Y
NEXT X
‘ Richiamo della routine
MATADD M1(), M2(), M3()
‘ Presentazione dei dati a video
FOR X=1 TO 10
        FOR Y=1 TO 10
                LOCATE X,Y*3
                PRINT USING “##”; M1(X, Y);
                LOCATE X,Y*3+40
                PRINT USING “##”; M2(X, Y);
                LOCATE X+12,Y*4+15
                PRINT USING “###”; M3(X, Y);
        NEXT Y
NEXT X
END
```

Questo programma va, naturalmente, compilato con il comando

BC PROMAT;

per ottenere il file PROMAT.OBJ necessario in seguito.
Il file MATADD.ASM contiene, invece, la routine scritta in Assembler che esegue la somma delle matrici. Questo è il seguente

```
; Routine MATADD - Versione 1.05. (MATADD.ASM)
; (da linkare con il programma in Quick Basic)
; A. Giuliana - Monreale 1991
;
; Viene usato il modello medio
.MODEL MEDIUM
; Inizio del segmento dei dati della routine
```

```
.DATA
  OLDDS        DW ?
  OLDES        DW ?
  HEADA        DW ?
  ARRASTR      DD ?
  ARRASTP      DD ?
  HEADB        DW ?
  ARRBSTR      DD ?
  ARRBSTP      DD ?
  HEADR        DW ?
  ARRRSTR      DD ?
  ARRRSTP      DD ?
; Inizio del segmento di codice della routine
; La routine viene dichiarata pubblica
.CODE
PUBLIC         MatAdd
MatAdd PROC
  ; Preparazione registri di segmento
  ; DS ---> area dati nella routine
  ; ES ---> area dati del Quick Basic
          mov bx,ds
          mov ax,@DATA
          mov ds,ax
          mov OLDDS,bx
          mov OLDES,es
          mov es,bx
  ; estrazione parametri
          pop ax
          pop dx
  ; Estrazione indirizzo della testata
  ; e dell'array risultato
          pop bx
          mov HEADR,bx
          mov di,offset ARRRSTR
          call GetArrAdd
  ; Estrazione indirizzo della testata
  ; e del secondo array da sommaredell'array risultato
          pop bx
          mov HEADB,bx
          mov di,offset ARRBSTR
          call GetArrAdd
  ; Estrazione indirizzo della testata
  ; e del primo array da sommare
          pop bx
          mov HEADA,bx
          mov di,offset ARRASTR
          call GetArrAdd
          push dx
```

```
        push ax
  ; Calcolo indirizzo di fine
  ; primo array da sommare
        mov bx,HEADA
        mov si,offset ARRASTR
        mov di,offset ARRASTP
        call CompArrEnd
  ; Calcolo indirizzo di fine
  ; secondo array da sommare
        mov bx,HEADB
        mov si,offset ARRBSTR
        mov di,offset ARRBSTP
        call CompArrEnd
  ; Calcolo indirizzo di fine
  ; array risultato
        mov bx,HEADR
        mov si,offset ARRRSTR
        mov di,offset ARRRSTP
        call CompArrEnd
  ; Ciclo di somma
  ; Diminuzione di 2 bytes
  ; di tutti i puntatori ad array
CycMin2:
        mov ax,word ptr ARRRSTP
        mov dx,word ptr ARRRSTP+2
        call Minus2
        mov word ptr ARRRSTP,ax
        mov word ptr ARRRSTP+2,dx
        mov ax,word ptr ARRBSTP
        mov dx,word ptr ARRBSTP+2
        call Minus2
        mov word ptr ARRBSTP,ax
        mov word ptr ARRBSTP+2,dx
        mov ax,word ptr ARRASTP
        mov dx,word ptr ARRASTP+2
        call Minus2
  ; Somma dei due elementi corrispondenti
  ; dei primi due arrays e deposito del
  ; risultato nel terzo array
        mov word ptr ARRASTP,ax
        mov word ptr ARRASTP+2,dx
        mov si,word ptr ARRASTP
        mov es,word ptr ARRASTP+2
        mov ax,es:[si]
        mov si,word ptr ARRBSTP
        mov es,word ptr ARRBSTP+2
        add ax,es:[si]
        jnc NoOvfl
```

```
  ; Risultato a 0 se non
  ; compreso nel range intero
            xor ax,ax
NoOvfl:
            mov di,word ptr ARRRSTP
            mov es,word ptr ARRRSTP+2
            mov es:[di],ax
  ; Il programma viene ripetuto solo
  ; se il puntatore attuale è diverso
  ; dall’indirizzo iniziale dell’array
            cmp di,word ptr ARRRSTR
            jnz CycMin2
            mov ax,word ptr ARRRSTP+2
            cmp ax,word ptr ARRRSTR+2
            jnz CycMin2
  ; Vengono recuperati i registri
  ; di segmento salvati all’inizio
  ; e termina la routine con il
  ; rientro al Quick Basic
            mov es,OLDES
            mov ds,OLDDS
            ret
MatAdd ENDP
  ; Routine per l’estrazione
  ; dalla testata dell’indirizzo
  ; di un array
GetArrAdd       PROC NEAR
            push es
            les cx,es:[bx]
            mov [di],cx
            mov [di+2],es
            pop es
            ret
GetArrAdd       ENDP
  ; Routine per il decremento
  ; di 2 unità da un dato a 32 bits
  ; indicante un indirizzo SEG:OFF
Minus2  PROC NEAR
            sub ax,2
            jnc NoPre
            sub dx,1000h
NoPre:
            ret
Minus2  ENDP
  ; Routine che calcola
  ; l’indirizzo finale
  ; di un array dalla testata
CompArrEnd      PROC NEAR
```

```
        xor ch,ch
        mov cl,es:[bx+8]
        add bx,0Eh
        mov ax,1
        xor dx,dx
NxtDim:
        mul word ptr es:[bx]
        add bx,4
        loop NxtDim
        shl ax,1
        rcl dx,1
        add ax,word ptr [si]
        adc dx,0
        mov [di],ax
        mov cl,0ch
        shl dx,cl
        adc dx,word ptr [si+2]
        mov [di+2],dx
        ret
CompArrEnd     ENDP
        END
```

Nell'esempio di cui sopra, si nota l'uso di routines assembler secondarie richiamate dalla routine assembler principale. Esse sono dichiarate di tipo NEAR perché comprese all'interno dello stesso segmento della principale.
Anche in questo caso, il tempo di esecuzione della routine è minimo rispetto a quello di una routine simile scritta in Quick Basic; anche in questo caso la routine assembler va compilata, con la riga

MASM MATADD;

e linkata con la riga

LINK PROMAT MATADD;

per realizzare il file PROMAT.EXE, eseguibile sotto MS-DOS.

Nel prossimo esempio, si può vedere come può essere realizzata una routine Assembler che visualizzi, molto velocemente, dati alfanumerici (solo in modo testo) a colori. La routine assembler è la seguente

```
; Routine VISUAL - Versione 1.25. (VISUAL.ASM)
; (da linkare con il programma in Quick Basic)
; A. Giuliana - Monreale 1991
;
; Viene usato il modello medio
.MODEL MEDIUM
; Inizio del segmento dei dati della routine
```

```
.DATA
  OLDDS       DW ?
  OLDES       DW ?
  STRING      DW ?
  LENSTR      DW ?
  VIDEO       DW ?
  COLOR       DB ?
  XCOO        DB ?
  YCOO        DB ?
; Inizio del segmento di codice della routine
; La routine viene dichiarata pubblica
.CODE
PUBLIC        Visual
Visual  PROC
  ; Preparazione registri di segmento
  ; DS ---> area dati nella routine
  ; ES ---> area dati del Quick Basic
        mov bx,ds
        mov ax,@DATA
        mov ds,ax
        mov OLDDS,bx
        mov OLDES,es
        mov es,bx
  ; Sono prelevati tutti i 4 parametri
  ; che sono depositati nel segmento Dati
        pop dx
        pop ax
        pop bx
        mov cx,es:[bx]
        mov bx,es:[bx+2]
        mov LENSTR,cx
        mov STRING,bx
        pop bx
        mov bx,es:[bx]
        mov COLOR,bl
        pop bx
        mov bx,es:[bx]
        mov YCOO,bl
        pop bx
        mov bx,es:[bx]
        mov XCOO,bl
        push ax
        push dx
  ; Viene ricavato nel registro si
  ; il segmento della memoria video (testo)
        push ds
        xor ax,ax
        mov ds,ax
```

```
            mov si,410h
            mov al,[si]
            pop ds
            and al,30h
            mov si,0B000h
            cmp al,30h
            jz Ok_ld
            mov si,0B800h
Ok_ld:
            mov VIDEO,si
  ; Non agisce se la stringa da
  ; visualizzare è nulla
            mov cx,LENSTR
            jcxz R_pro
  ; Calcola l'offset su memoria video
  ; del primo carattere della stringa
  ; e lo deposita nel registro di
            push cx
            xor ah,ah
            mov al,YCOO
            dec ax
            mov cx,160
            mul cx
            xor ch,ch
            mov cl,XCOO
            mov di,cx
            dec di
            shl di,1
            add di,ax
            pop cx
  ; Trasferisce la stringa dalla memoria
  ; del QB alla memoria video
            mov si,STRING
            push ds
            push es
            mov ax,VIDEO
            mov es,ax
            mov ah,COLOR
            pop ds
Newch:
            mov al,[si]
            mov es:[di],al
            mov es:[di+1],ah
            add di,2
            inc si
            loop Newch
  ; Riprende i registri salvati
  ; e termina la routine
```

```
        pop ds
R_pro:
        mov es,OLDES
        mov ds,OLDDS
        ret
Visual          ENDP
        END
```

Il programma di prova, scritto in QB, per tale routine, potrebbe essere, ad esempio, il seguente

```
' (Main Module PROVIS.BAS)
DEFINT A-Z
DECLARE SUB VISUAL(XC, YV, COL, VSTR$)
CLS
V$="Quick Basic - "
V$=V$+"Tecniche avanzate di programmazione"
FOR Y=1 TO 25
      VISUAL 15, Y, RND*15, V$
NEXT Y
END
```

Per quest'ultimo non c'è bisogna di commento. Resta da precisare che il programma in QB va compilato con

BC PROVIS;

quello in assembler, con

MASM VISUAL;

ed i due vanno linkati con

LINK PROVIS VISUAL;

per ottenere il file PROVIS.EXE.

Come ultimo esempio, viene riportata la seguente routine assembler che permette di cambiare segno ad un valore numerico intero lungo che le viene passato come parametro. Questo esempio è importante, dato che, in questo caso, la routine diventa una nuova **funzione** del linguaggio.

```
; Routine LNEG - Versione 1.5. (LNEG.ASM)
; (da linkare con il programma in Quick Basic)
; A. Giuliana - Monreale 1991
;
; Viene usato il modello medio
.MODEL MEDIUM
; Inizio del segmento dei dati della routine
```

```
.DATA
    OLDROFF        DW ?
    OLDRSEG        DW ?
; Inizio del segmento di codice della routine
; La routine viene dichiarata pubblica
.CODE
PUBLIC             LNeg
LNeg        PROC
    ; Preleva il parametro (intero lungo)
    ; nei registri DI:SI (32 bits H-L)
            pop OLDROFF
            pop OLDRSEG
            pop bx
            mov si,es:[bx]
            mov di,es:[bx+2]
            push OLDRSEG
            push OLDROFF
    ; Esegue la sottrazione da 0
    ; del parametro; risultato
    ; a 32 bit in DX:AX
            xor ax,ax
            xor dx,dx
            sub ax,si
            sbb dx,di
    ; Ritorno al QB con il
    ; valore della dunzione
    ; in DX:AX (32 bits)
            Ret
LNeg        ENDP
    END
```

Il programma di prova della routine suddetta, è il seguente

```
' (Main Module PROLNEG.BAS)
DECLARE FUNCTION LNEG&(VALUE&)
CLS
INPUT "Valore : ", V&
INPUT "Valore negato : ", LNEG&(V&)
END
```

Notare che la routine LNEG è stata dichiarata come **FUNCTION** (in quanto viene ritornato un valore), di tipo intero lungo (il simbolo & alla fine del nome indica che il valore ritornato è a 32 bits in DX:AX). Anche il parametro naturalmente, è indicato come intero lungo.
L'esempio precedente va compilato e linkato con le righe seguenti

BC PROLNEG;
MASM LNEG;
LINK PROLNEG LNEG;

Se la routine assembler dovesse ritornare un dato diverso da un intero o intero lungo (ad esempio, singola o doppia precisione o stringa), si dovrebbe fornire l'indirizzo (solo offset) dello stesso nel registro AX prima di tornare al QB). Il seguente esempio mostra come è possibile fare ciò con una stringa. Notare che il dato da ritornare è allocato (ultimo parametro) come stringa alla chiamata della funzione

```
; Routine InvStr - (INVSTR.ASM)
; (da linkare con il programma in Quick Basic)
; A. Giuliana - Monreale 1991
;
; Viene usato il modello medio
.MODEL MEDIUM
; Inizio del segmento dei dati della routine
.DATA
        OLDROFF     DW ?
        OLDRSEG     DW ?
        STR1        DW ?
        LENSTR      DW ?
        STR2        DW ?
        RES         DW ?
; Inizio del segmento di codice della routine
; La routine viene dichiarata pubblica
.CODE
PUBLIC              InvStr
InvStr      PROC
        ; Vengono prelevati i parametri
                pop OLDROFF
                pop OLDRSEG
                pop bx
                mov RES,bx
                mov ax,[bx+2]
                mov STR2,ax
                pop bx
                mov ax,[bx]
                mov LENSTR,ax
                mov ax,[bx+2]
                mov STR1,ax
                push OLDRSEG
                push OLDROFF
        ; Non opera con le stringhe nulle
                mov cx,LENSTR
                jcxz Done
        ; Trasferisce la stringa al contrario
                mov si,STR1
                mov di,STR2
                add di,LENSTR
```

(… da completare a cura del lettore …)

Il programmino in QB usato per provare tale funzione, è il seguente

```
' (Main Module PROINVST.BAS)
DECLARE FUNCTION INVSTR$(X$, T$)
CLS
INPUT "String : ", XX$
TT$=SPACE$(LEN(XX$))
PRINT "Inverse String : "; INVSTR$(XX$, TT$)
END
```

in cui è importante notare che la riga evidenziata è necessaria per allocare spazio per la stringa in cui verrà depositato il risultato dell'operazione.
I programmi sono compilati e linkatu con le righe

```
BC PROINVST;
MASM INVSTR;
LINK PROINVST INVSTR;
```

Per provare il programma, basta, a questo punto, lanciare sotto MS-DOS il file PROINVST.EXE.

7.1.4 Interfacciamento al MSC Compiler v. 6.0

L'interfacciamento del Quick Basic al compilatore C, v. 6.0. della Microsoft, non presenta particolari difficoltà, tuttavia è necessario rispettare alcune regole per potere realizzare programmi con parti scritte nei due linguaggi.

Dato che all'interno di un modulo scritto in C possono esistere soltanto funzioni, la dichiarazione in QB di una SUB esterna a cui corrisponde una funzione scritta in C è possibile solo se a tale funzione viene preposta la parola chiave **void**. Se in QB viene dichiarata una FUNCTION non è previsto, naturalmente, l'uso di tale parola chiave.

La frase DECLARE, in QB, è il mezzo con cui si dichiara al compilatore, come dovrà essere collegata la funzione esterna scritta in C. La sua sintassi (v. 5.1.1) prevede le seguenti parti

- **parola SUB o FUNCTION** - si usa la prima se non viene ritornato valore dalla funzione C; in questo caso la funzione deve essere di tipo void. Si usa la seconda quando la funzione C ritorna un valore;

- **nome della procedura** - il nome della procedura deve essere quello usato nella funzione C; nel caso in cui nel nome di quest'ultima compaia il carattere underscore (_), non consentito in BASIC, si usa l'opzione ALIAS, oppure l'opzione CDECL ed un punto al posto dell'underscore;

- **opzione CDECL** - specificando questa opzione, viene automaticamente aggiunto un underscore all'inizio del nome della SUB o FUNCTION per renderlo compatibile con il linguaggio C. Inoltre, il passaggio dei parametri viene fatto come impone il linguaggio C;

- **opzione ALIAS "alias"** - nel caso in cui il nome della funzione in C e quello della SUB o FUNCTION in QB, differiscano, viene specificato, tra virgolette, quello adottato nel modulo C;

- **parametri** - vengono specificati, tra parentesi, tutti i parametri da passare al modulo C. È consigliabile usare l'opzione **BYVAL** dato che, normalmente, i dati sono passati per indirizzo al modulo C.

Ad esempio, se si dovesse dichiarare nel modulo scritto in QB una SUB, con un parametro, la cui funzione, di tipo void in C avesse il nome **Fine_Tx**, questa dovrebbe essere la frase DECLARE da usare

DECLARE SUB FineTx CDECL ALIAS "Fine_Tx" (BYVAL Par1%)

* <u>Passaggio dei parametri</u>

Per default, il QB passa i parametri per indirizzo vicino (Near Reference) e cioè passa soltanto l'offset relativo al dato considerato. Tuttavia è possibile passare i dati per valore usando l'opzione BYVAL nella DECLARE. Questo è il metodo con cui il linguaggio C si aspetta che normalmente vengano passati tutti i dati **che non sono array o di tipo composto**; per quest'ultimi, esso si attende un puntatore a tali dati (Near Reference).

Nel caso in cui si dovessero passare alla funzione C dei dati tramite il loro indirizzo completo di segmento e offset (Far Reference), si dovrebbe usare l'opzione SEG nella DECLARE o l'istruzione CALLS nella chiamata durante il programma in QB.

* <u>Stringhe</u>

Il passaggio delle stringhe dal QB al C avviene usando la funzione SADD che consente di ottenere l'indirizzo effettivo dell'area di memoria che contiene la stringa stessa. La funzione LEN, inoltre, può essere usata per passare alla funzione C la lunghezza, in caratteri, della stringa stessa. Notare che ambedue i dati devono essere passati per valore (opzione BYVAL nella DECLARE).

* <u>Compilazione del modulo C e linkaggio</u>

Il file con il programma C non deve contenere la funzione main() e deve essere compilato usando l'opzione /c per generare solo il file oggetto e non l'eseguibile. Inoltre è necessaria l'opzione /Mn per specificare il modello (medio) che il compilatore C deve usare, e l'opzione /Gs per evitare che vengano incluse delle routines di controllo dello stack, da parte del compilatore C, che potrebbero pregiudicare il funzionamento dell'eseguibile.

Quest'ultimo si ottiene con il LINK del modulo BASIC compilato (specificato per primo nella linea di comando) e del modulo oggetto del programma C.
È necessaria l'opzione /NOE per evitare che vengano segnalate delle routines o simboli dichiarati più volte (effetto quasi scontato dell'uso di più librerie simili), ed è cosa normale specificare la libreria C che deve essere usata (normalmente la MLIBCE.LIB).

7.1.5 Esempi di interfacciamento al MSC Compiler v. 6.0

Nel primo esempio di interfacciamento tra programmi nei due linguaggi, si intende usare la funzione di libreria del compilatore C che calcola il logaritmo in base 10 di un valore numerico; tale funzione, seppure ottenibile con una formula di trasformazione del logaritmo dalla base *e*, è disponibile ed immediatamente utilizzabile realizzando l'interfaccia con un programma C.
Il programma, scritto in QB, tramite il quale viene provata tale funzione, è il seguente

```
' (Main Module LOG10.BAS)
'
' La funzione LogBase10 di tipo double
' è dichiarata per il linguaggio C
' ed usa un solo parametro double
' passato per valore
DECLARE FUNCTION LogBase10# CDECL (BYVAL LVal#)
CLS
INPUT "Value : "; Value#
' La funzione viene usata come se fosse standard
PRINT LOG(Value#), LogBase10#(Value#)
END
```

mentre il file contenente il modulo C, è quest'altro

```
/* Modulo LogB10.C */
#include <math.h>
/* dichiarazione della funzione */
Double LogBase10(double v)
{
      /* La funzione ritorna il valore del
         Logaritmo in base 10 del parametro */
      return(log10(v));
}
```

Il programma in QB va compilato con

BC LOG10;

e quello in C con

CL /c /Mm /Gs logb10.c

Infine, i due moduli oggetto (file LOG10.OBJ e LOGB10.OBJ) sono collegati con il comando

LINK /NOE LOG10 LOGB10;

Il file eseguibile LOG10.EXE così ottenuto, è quindi utilizzabile sotto MS-DOS.

Il secondo esempio mostra come è possibile usare la funzione **qsort** della libreria del compilatore C che, in modo semplice, permette l'ordinamento rapido di dati. Il codice che segue mostra come tale funzione possa essere usata per operare su array di interi definiti in un modulo scritto in QB

```
' (Main Module PROSORT.BAS)
DEFINT A-Z
DECLARE SUB ISort CDECL (BYVAL ArrayAdd, BYVAL Nels)
RANDOMIZE TIMER
CLS
MaxEL = 200
DIM X(1 TO 20000)
LOCATE 1, 1
FOR XX=1 TO MaxEl
        X(XX) = RND * 1000
        PRINT X(XX);
        IF INKEY$ = CHR$(27) THEN EXIT FOR
NEXT XX
PRINT
ArrAdd=VARPTR(X(1))
ISort ArrAdd, MaxEl
LOCATE 1, 1
FOR XX=1 TO MaxEl
        PRINT XX(X);
        IF INKEY$ = CHR$(27) THEN EXIT FOR
NEXT XX
END
```

Porre attenzione al fatto che la routine ISort, dichiarata con l'istruzione DECLARE, viene definita, tramite l'opzione CDECL, in maniera da renderla compatibile con il linguaggio C e le sue convenzioni di chiamata. Notare inoltre che, come normalmente per il linguaggio C, i parametri sono passati per valore e che il valore del primo parametro è l'indirizzo dell'array; non sarebbe stata la stessa cosa passare il primo parametro per indirizzo dato che l'indirizzo passato sarebbe stato quello della **testata** dell'array (in cui sono specificati l'indirizzo effettivo, il numero delle dimensioni e le grandezze di ognuna di queste) e non quello dell'array

```
#include <search.h>
#include <string.h>
void isort(int *, unsigned);
int icmp(int *, int *);
void isort(int *array, unsigned nels)
{
  qsort(array, nels, sizeof(int), icmp);
}
int icmp(int *, int *)
{
  return(*n1 - *n2);
}
```

Nel testo scritto in C è importante invece vedere che l'indirizzo dell'array (il primo parametro) viene inteso proprio come **puntatore ad intero** mentre il secondo come semplice valore senza segno.
L'esempio suddetto va compilato con le righe seguenti

BC PROSORT;

per il file in QB

CL -c –Gs –Mm isort.c

per il file in C e va linkato con

LINK /NOE PROSORT ISORT;

Il file PROSORT.EXE può essere così eseguito. Attenzione, nel provare questo programma, a non tentare ordinamenti con un numero di elementi molto grande perché si deve tenere conto di due fatti

1) se si usa un array dinamico bisogna fare in modo che la funzione in C conosca il segmento al quale esso si trova;
2) dato che la routine di quick sort usa intensamente lo stack, può capitare che, per un numero elevato di elementi da ordinare, la capacità dello stack stesso venga superata durante l'ordinamento che non può così essere completato.

Il prossimo esempio mostra come possa essere utilizzata una funzione di libreria del compilatore C Microsoft, la **movedata**, che permette la copia di porzioni di memoria anche tra segmenti diversi. Tale routine, inclusa nella libreria BPLUS, ed usata nel programma dimostrativo conclusivo (QDOS), viene usata nelle prossime routines per copiare, in modo veloce, il contenuto di un array di interi in un altro. Per effettuare una dimostrazione tangibile della differenza di velocità di esecuzione tra un programma che usi la funzione movedata, ed un altro che sfrutti le sole istruzioni del Quick Basic, viene mostrato un programma di prova che effettua **per cento volte consecutive** la copia di un array di interi in un altro eguale, entrambi composto da **32000 elementi**.
La funzione C che permette l'interfaccia con la movedata, è stata chiamata **mvdata** ed il, breve, testo relativo è il seguente

```
#include <memory.h>
#include <dos.h>
void mvdata(sseg, soff, dseg, doff, siz)
{
        movedata(sseg, soff, dseg, doff, siz);
        return;
}
```

Come si può vedere, la funzione è stata dichiarata di tipo **void** (non viene ritornato alcun valore alla routine chiamante, cosicché, in Quick Basic, essa dovrà essere dichiarata come una SUB), ed ammette cinque parametri, che sono, nell'ordine

1) il valore del segmento della zona di memoria di origine (valore intero);
2) il valore dell'offset della zona di memoria di origine (valore intero);
3) il valore del segmento della zona di memoria di destinazione (valore intero);
4) il valore dell'offset della zona di memoria di destinazione (valore intero);
5) il numero dei bytes che devono essere copiati (valore intero senza segno; fino ad un massimo di 65535).

Il seguente testo scritto in Quick Basic è, invece, quello che permette di usare la routine precedente e che mostra la differenza di tempo di esecuzione usando i due metodi diversi

```
' (Main Module ARRCPY.BAS)
DEFINT A-Z
DECLARE SUB MVDATA CDECL (BYVAL Sse, BYVAL Sof, BYVAL Dse, BYVAL Dof, BYVAL Sz)
CONST Siz=32000
DIM M1(1 TO Siz), M2(1 TO Siz)
CLS
' L'array origine (M1) viene riempito con dati numerici per
' controllare la funzionalità del programma
FOR X=1 TO Siz
        M1(X)=X
NEXT X
' Inizia la sequenza di copia dell'array che sfrutta la routine
' MVDATA. Il tempo impiegato è di circa 3.4 secondi
K#=TIMER
FOR TIM=1 TO 100
        S&=UBOUND(M1)*2&
IF S&>32767 THEN
                S=S& - 65536
ELSE
                S=S&
END IF
MVDATA VARSEG(M1(1)), VARPTR(M1(1)), VARSEG(M2(1)), VARPTR(M2(1)), S
NEXT TIM
PRINT TIMER – K#
' Qui inizia la parte scritta in Quick Basic per la copia
' dell'array. Essa impiega più di 29 secondi
K#=TIMER
FOR TIM=1 TO 100
        FOR X=1 TO Siz
                M2(X)=M1(X)
        NEXT X
NEXT TIM
PRINT TIMER – K#
' Questa parte di programma è inclusa solo per controllare
```

```
' la riuscita della copia dell'array
FOR X=1 TO 100
        PRINT M2(X);
NEXT X
PRINT "…";
FOR X=Siz-100 TO Siz
        PRINT M2(X);
NEXT X
END
```

Ovviamente, per controllare la funzionalità di uno dei due metodi, senza usare l'altro, è sufficiente far precedere le linee di programma non richieste da delle frasi REM (o con il carattere ‘).
La routine MVDATA va compilata con la riga

CL /c /Mm /Gs mvdata.c

per ottenere così il file MVDATA.OBJ; la routine di prova scritta in QB va compilata, invece, con la riga

BC ARRCPY;

ed il file ARRCPY.OBJ così ottenuto va linkato al precedente, con la riga di comando

LINK ARRCPY MVDATA;

Il file ARRCPY.EXE ottenuto tramite il suddetto comando, è eseguibile sotto MS-DOS.

7.2 BPLUS LIBRARY

7.2.1 La libreria BPLUS

La libreria presentata a conclusione del libro, non è altro che una collezione di routines, più o meno utili, che comunque dimostrano nella maniera più completa possibile, tutte le possibilità di realizzazione di moduli pluri-linguaggio con il Quick Basic.
In essa è incluso un sistema di gestione di finestre testo con dei comandi per la loro apertura, la visualizzazione dei dati, l'input e la chiusura; le routines corrispondenti sono le seguenti

CENTREWIN (OpWin, Astr$, Col, Ret, Row)
CINPUT (Px, Py, Lun, OldS$, NewS$, Col, Typ, Prompt)
CLEARBOX (Xt, Yt, Xb, Yb, Col)
CLEARWINDOW (OpWin, Col)
CLOSEWINDOW (OpWin)
COLORWINDOW (OpWin, Col)
CURSORWINDOW (OpWin, Row, Col)
OPENWINDOW (Xt, Yt, Xb, Yb, Col, Crn)
PRINTWINDOW (OpWin, Astr$, Col, Ret)

È stato incluso anche, un sistema di gestione di popup menu, le cui routines sono le seguenti

CLOSEPOPMENU (Menu, Handle$)
GETKEYMENU (Menu)
OPENPOPMENU (Menu, Handle$)
RESTBOX (Xt, Yt, Xb, Yb, Handle$)
SAVEBOX (Xt, Yt, Xb, Yb, Handle$)
SELECTOPZMENU! (MaxMenus, StMenu)
SETPOPMENU (MaxMenus)

ed un sistema di controllo del mouse, le cui funzioni sono

GETMOUSEBUTTON
GETMOUSEPOS (POSX, POSY)
MOUSE (OPER, COOX, COOY, BUTTON)
MOUSECHECK
MOUSEOFF
MOUSEON
SETMOUSEPOS (POSX, POSY)

Sono state incluse, poi, moltre altre routines, il cui elenco è precisato in seguito, per la gestione di file, tastiera, video e memoria che permettono la realizzazione di programmi molto più curati dal punto di vista estetico e di più semplice realizzazione rispetto al normale.

Le routines sono raccolte all'interno di un file, chiamato BPLUS.BAS, che va compilato con il comando

BC BPLUS;

per ottenere il file BPLUS.OBJ, dal quale è possibile realizzare la libreria in questione. A tal riguardo, si deve usare il programma LIB della Microsoft, avendo cura di includere anche i moduli oggetto dei programmi MVDATA, EXPEXT ed INTRPT realizzati con altri linguaggi ed ottenuti tramite i compilatori relativi. In definitiva, questo è l'elenco dei file e una breve descrizione del loro contenuto, che sono necessari per la realizzazione della BPLUS.LIB

BPLUS.INC file include contenente tutte le dichiarazioni delle routines della libreria BPLUS con i relativi parametri, ad eccezione delle INTERRUPT e INTERRUPTX; esse sono elencate in ordine alfabetico solo per comodità (v. App. B);

TYPES.INC file include contenente tutti i dati di tipo utente usati dalle routines della libreria BPLUS; in esso compaiono i tipi usati per la gestione delle finestre, per la gestione della directory e per le funzioni INTERRUPT ed INTERRUPTX (v. App. B);

COMMON.INC file include contenente tutte le frasi COMMON per la dichiarazione degli arrays e delle variabili condivise tra tutti i moduli; si ricorda, infatti, che lavorando con una libreria esterna, tutti i programmi che la usano sono considerati come appartenenti ad un altro modulo (v. App. B);

DOS.INC file include contenente le dichiarazioni delle routines INTERRUPT ed INTERRUPTX (v. App. B);

POPMENU.INC file include contenente le indicazioni necessarie per la preparazione dei dati utilizzati dalle routines di gestione dei popup menus (v. App. B);

WIN.INC file include relativo alla dichiarazione dei dati utilizzati dalle routines di gestione delle finestre testo (v. App. B);

DIR.INC file include contenente le dichiarazioni dei dati relativi alla gestione delle directory (v. App. B);

DATETIME.INC file include contenente le dichiarazioni dei dati relativi alla gestione di date e orari (v. App. B);

BPLUS.BAS file sorgente, scritto in Quick Basic, contenente tutte le routines della libreria tranne quelle scritte in C o Assembler;

BPLUS.OBJ file oggetto ottenuto dalla compilazione del precedente, tramite il comando BC BPLUS;

MVDATA.C file sorgente, scritto in C, contenente la routine per il trasferimento di una zona di memoria qualsiasi (v. App. B);

MVDATA.OBJ file oggetto ottenuto dalla compilazione del precedente, tramite il comando CL /c /Mm /Gs mvdata.c;

EXPEXT.ASM file sorgente, scritto in Assembler, contenente la routine per il controllo della quantità di memoria espansa ed estesa a disposizione del sistema (v. App. B);

EXPEXT.OBJ file oggetto ottenuto dalla compilazione del precedente, tramite il comando MASM EXPEXT;

INTRPT.ASM file sorgente, scritto in Assembler, contenente le due funzioni, INTERRUPT ed INTERRUPTX della Microsoft per l'esecuzione di un qualsiasi interrupt di sistema; **questo file deve essere corretto come mostrato in seguito (v. 7.2.2)**;

INTRPT.OBJ file oggetto ottenuto dalla compilazione del precedente, tramite il comando MASM INTRPT;

BPLUS.LIB file di libreria contenente tutte le routines precedenti, ottenuto con la riga LIB BPLUS.LIB BPLUS MVDATA EXPEXT INTRPT;

BPLUS.QLB file di libreria (per l'ambiente) ottenuto dal precedente tramite il comando LINK /Q BPLUS.LIB,BPLUS.QLB,NUL,BQLB40.LIB MLIBCE.LIB; (il file MLIBCE.LIB è la libreria ottenuta all'installazione del compilatore C tramite l'unione delle varie librerie fornite con lo stesso per il modello medio di indirizzamento della memoria).

Porre attenzione al fatto che le routines non sono state scritte con tutti i controlli delle situazioni di errore che si potrebbero presentare tramite un loro cattivo utilizzo. Per questo motivo si raccomanda di usarle in modo appropriato o di aggiungere tali controlli nel programma che li utilizza. Inoltre, notare che, con l'uso della libreria BPLUS.QLB con l'ambiente QB, si possono presentare delle difficoltà per la complessa gestione della memoria a cui il sistema è sottoposto; infatti la memoria allocata da molte routines (gestione finestre e popup menus) non è sempre rilasciata in modo corretto e ciò può fare 'bloccare' il sistema stesso; si consiglia quindi, di compilare ed eseguire i programmi sorgenti usando la BPLUS.LIB. A conclusione di tale precisazione, considerare quindi che tale libreria, raccoglie delle routines sperimentali, che non solo è possibile migliorare, ma che si spera il lettore possa anche ampliare.

7.2.2 Routines in Quick Basic commentate

La maggior parte delle routines della libreria BPLUS sono scritte in Quick Basic, anche se, alcune di queste ne sfruttano altre scritte in Assembler o in C. In particolare, sono usate le due funzioni INTERRUPT ed INTERRUPTX incluse nel pacchetto del compilatore Quick Basic; queste due funzioni permettono di eseguire un qualsiasi interrupt di sistema preparand, in precedenza, tutti i registri coinvolti e restituendoli, alla fine, con i valori di ritorno.
Purtroppo, per la versione 4.0., bisogna notare che tale routines presentano un piccolo errore che però ne pregiudica, a volte, il funzionamento. Fortunatamente, viene fornito anche il testo sorgente scritto in Assembler che permette di correggere tale errore ed ottenere un modulo oggetto corretto. Tale errore è presente nella riga numero 49 del sorgente (file INTRPT.ASM) che appare la seguente

```
INT_DI  =     -1EH          ;INT DI register value
```

e che deve essere, evidentemente, corretta nella seguente

```
INT_DI  =     -0EH          ;INT DI register value
```

Dopo tale correzione, essa può essere ricompilata con il MASM v. 5.0., con il seguente comando

```
MASM INTRPT;
```

e le librerie che la utilizzano (QB.LIB e QB.QLB) possono essere ricostruite con i comandi

```
LIB QB -+INTRPT;
LINK /Q QB.LIB,QB.QLB,NUL,BQLB40.LIB;
```

Naturalmente, il file di riserva, QB.BAK, può essere, a questo punto, eliminato.

Il file INTRPT.OBJ corretto, ottenuto in precedenza, deve essere incluso nella libreria BPLUS.LIB per evitare di doverlo sempre specificare ad ogni compilazione di un programma che sfrutti le routines suddette. La realizzazione di tale operazione è stata, comunque, descritta in precedenza.

Segue adesso, una sorta di 'reference' delle routines scritte in QB, elencate in ordine alfabetico con le relative informazioni necessarie alla comprensione del loro funzionamento e ad una loro corretta utilizzazione.

Nome : **ABEEP**

Permette l'emissione di un suono tramite l'altoparlante del sistema, solo se la variabile globale di nome **BeepYesNo** ha valore logico vero (diverso da zero).

```
SUB ABEEP
	IF BeepYesNo = -1 THEN
		BEEP
	END IF
END SUB
```

Nome : **ADDZERO2STR$**

Tale funzione restituisce una stringa contenente il valore numerico passato come primo parametro ma di lunghezza, in caratteri, specificata dal secondo. Vengono aggiunti degli zeri iniziali se necessari.

```
FUNCTION ADDZERO2STR$(X AS SINGLE, CF AS INTEGER)
	ADDZERO2STR$=STRING$(CF-LEN(STR$(X))+1,48) + LTRIM$(STR$(X))
END FUNCTION
```

Nome : **BEEPNO**

Pone la variabile globale di nome **BeepYesNo** eguale al valore logico falso (zero). In tal modo non verrà emesso alcun suono se si usasse la routine ABEEP.

```
SUB BEEPNO
	BeepYesNo = 0
END SUB
```

Nome : **BEEPYES**

Pone la variabile globale di nome **BeepYesNo** eguale al valore logico vero (diverso da zero). Così verrà emesso un suono al momento dell'uso della routine ABEEP.

```
SUB BEEPYES
	BeepYesNo = -1
END SUB
```

Nome : **CAPSOFF**

Provvede ad azzerare il bit 6 del byte contenuto nella cella di indirizzo 0040:0017 esadecimale. Così il S.O. provvede a disabilitare il tasto Caps-Lock, spegnere la relativa spia sulla tastiera e si predispone per operare con le lettere minuscole.

```
SUB CAPSOFF STATIC
	DEF SEG=&H40
	POKE &H17, PEEK(&H17) AND &HBF
	DEF SEG
```

```
END SUB
```

Nome : **CAPSON**

Pone ad 1 il bit 6 del byte contenuto nella cella di indirizzo 0040:0017 esadecimale. Il S.O. provvede quindi, ad abilitare il tasto Caps-Lock, accendere la relativa spia sulla tastiera e si predispone per operare con le lettere maiuscole.

```
SUB CAPSON STATIC
	DEF SEG=&H40
	POKE &H17, PEEK(&H17) OR &H40
	DEF SEG
END SUB
```

Nome : **CENTREWIN**

È una delle funzioni di gestione delle finestre. Essa è utilizzata per visualizzare in una riga di una finestra, una stringa in maniera centrata. I parametri accettati dalla funzione sono, nell'ordine

- il numero della finestra ritornato dalla OPENWINDOW;
- la stringa da visualizzare;
- il codice del colore da utilizzare;
- il codice Return da utilizzare (vedere PRINTWINDOW);
- il numero della riga prescelta.

L'ultimo parametro non è assoluto, ma relativo alle coordinate della finestra. La funzione ritorna un codice d'errore o il valore 0 per indicare che l'operazione si è conclusa con successo.

```
DEFINT A-Z
FUNCTION CENTREWIN (OpWin, Astr$, Col, Ret, Row)
	IF OpWin<1 OR OpWin>MAXW THEN
		CENTREWIN=-4			' Invalid Window Number
		EXIT FUNCTION
	END IF
	IF Windows(OpWin).Used = 0 THEN
		CENTREWIN=-5			' No Window Open
		EXIT FUNCTION
	END IF
	IF LEN(Astr$) = 0 THEN
		EXIT FUNCTION
	END IF
	Colu = Windows(OpWin).Cols / 2 - LEN(Astr$) / 2 + 1
	IF Colu<1 THEN
		EXIT FUNCTION
	END IF
	DUM = CURSORWINDOW(OpWin, Row, Colu)
	DUM = PRINTWINDOW(OpWin, Astr$, Col, Ret)
	CENTREWIN = 0
END FUNCTION
```

Nome : **CINPUT**

Si tratta della classica funzione di input controllato che permette l'immissione di dati, sia alfanumerici che numerici, in maniera più efficiente rispetto a quella fornita da istruzioni quali INPUT o LINE INPUT.

Essa ammette ben 8 parametri, che sono, nell'ordine

- valore della coordinata x (numero colonna) dello schermo dal cui punto partirà l'input (è assoluta rispetto allo schermo);
- valore della coordinata y (numero riga) dello schermo dal cui punto partirà l'input (è assoluta rispetto allo schermo);
- lunghezza massima, in caratteri del dato in input (non può essere zero);
- stringa rappresentante il dato prima dell'input; questo parametro è utile quando si deve modificare un dato già posseduto mentre va posto eguale a stringa nulla, quando l'input si riferisce ad un dato nuovo;
- variabile stringa nella quale viene depositato il dato alla fine dell'input;
- colore usato nell'input del dato;
- tipo di dato: 0 = alfanumerico, 1 = numerico;
- codice ASCII del carattere usato come prompt; se non si desidera, si può usare il 32.

La funzione ritorna un valore numerico codificato nel seguente modo

-1 si è verificato un errore durante l'input numerico;
0 si è concluso l'input con il tasto ESC;
1 si è concluso l'input con il tasto RETURN.

```
FUNCTION CINPUT (Px, Py, Lun, OldS$, NewS$, Col, Typ, Prompt)
    Ins=0
    COLOR Col MOD 16, Col \ 16
    IF OldS$ <> “” THEN
        OldS$ = LEFT$(OldS$, Lun)
    END IF
    LOCATE Py, Px
    PRINT OldS$; STRING$(Lun-LEN(OldS$), Prompt);
    SOut$ = LEFT$(OldS$, Lun)
    ActX = Px + LEN(SOut$)
    Idx = LEN(SOut$) + 1
    DO
        LOCATE Py, ActX, 1
        Ak$ = INKEY$
        Akl = LEN(Ak$)
        IF Akl = 1 THEN
            AkCh = ASC(Ak$)
            SELECT CASE AkCh
                CASE 8
                    IF Idx>1 THEN
                        ActX=ActX-1
                        Idx=Idx-1
                        LOCATE , ActX
```

```
                    PRINT STRING$(1 , Prompt);
                    IF Idx=LEN(SOut$) THEN
                        SOut$=LEFT$(SOut$, LEN(SOut$)-1)
                    ELSE
                        MID$(SOut$, Idx, 1) = " "
                    END IF
                ELSE
                    SOUND 500, 1
                END IF
            CASE 13
                CINPUT = 1
                EXIT DO
            CASE 27
                CINPUT = 0
                EXIT DO
            CASE IS > 31
              IF Typ=0 OR (Typ=1 AND ISDIGIT(AkCh)) THEN
                IF Idx <= Lun THEN
                  PRINT Ak$;
                  IF Ins THEN
                    PRINT MID$(SOut$, Idx, Lun-Idx);
                  END IF
                  ActX=ActX+1
                  IF Idx=LEN(SOut$)+1 THEN
                    SOut$= SOut$+Ak$
                  ELSE
                    IF Ins THEN
                      SOut$=LEFT$(SOut$,Idx-1)+Ak$+MID$(SOut$,Idx, Lun-Idx)
                    ELSE
                     MID$(SOut$, Idx, 1) = Ak$
                    END IF
                  END IF
                  Idx=Idx+1
                ELSE
                  SOUND 500,1
                END IF
              ELSE
                BEEP
              END IF
            CASE ELSE
        END SELECT
    END IF
    IF Akl = 2 THEN
        AkCh = ASC(RIGHT$(Ak$, 1))
        SELECT CASE AkCh
            CASE 71
                Idx=1
                ActX=Px
```

```
                    CASE 79
                        Idx=LEN(SOut$)+1
                        ActX=Px+Idx-1
                    CASE 75
                        IF Idx>1 THEN
                            Idx=Idx-1
                            ActX=ActX-1
                        END IF
                    CASE 77
                        IF Idx<LEN(SOut$) THEN
                            Idx=Idx+1
                            ActX=ActX+1
                        END IF
                    CASE 82
                        Ins = Ins XOR -1
                        IF Ins THEN
                            LOCATE , , 1, 1, 12
                        ELSE
                            LOCATE , , 1, 13, 13
                        END IF
                    CASE 83
                        IF Idx <= LEN(SOut$) THEN
                            SOut$=LEFT$(SOut$, Idx-1)+MID$(SOut$, Idx+1)
                            OldX= POS(0)
                            LOCATE , Px
                            PRINT SOut$; STRING$(Lun-LEN(SOut$), Prompt);
                            LOCATE , OldX
                        ELSE
                            BEEP
                        END IF
                    CASE ELSE
                END SELECT
        END IF
LOOP
IF Typ=1 THEN
        PoPos = INSTR$(SOut$, ".")
        NDec = LEN(SOut$)-PoPos
        IF PoPos=0 THEN
                NDec=0
        END IF
        Tv#=VAL(SOut$)
        LOCATE , Px
        SOut$ = STR$(Tv#)
        OPoPos=INSTR(SOut$, ".")
        IF OPoPos>0 THEN
                SOut$=LEFT$(SOut$, OPoPos+NDec)
        END IF
        IF (LEN(SOut$) <= Lun) OR NDec>0 THEN
```

```
                SOut$=LEFT$(SOut$, Lun)
                IF RIGHT$(SOut$, 1) = "." THEN
                    SOut$=LEFT$(SOut$, Lun-1)
                END IF
                PRINT SOut$; STRING$(Lun-LEN(SOut$), Prompt);
            ELSE
                SOut$=STRING$(Lun, "*")
                CINPUT=-1
            END IF
        END IF
        New$=SOut$
        LOCATE , , 0, 13, 13
END FUNCTION
```

Nome : **CLEARBOX**

È una routine usata per cancellare il contenuto di una finestra video qualsiasi. I parametri di tale routine sono costituiti dalle coordinate della zona video prescelta (decrementati tutti come per la SCROLLUPBOX) e dal colore prescelto per la finestra. Essa usa la routine SCROLLDNBOX con un numero di linee eguale a zero.

```
SUB CLEARBOX (Xt, Yt, Xb, Yb, Col)
        SCROLLUPBOX Xt, Yt, Xb, Yb, 0, Col
END SUB
```

Nome : **CLEARWINDOW**

È la funzione che pemette di cancellare una sola finestra aperta in precedenza. Essa ammette due parametri che sono

- il numero della finestra su cui operare ritornato dalla OPENWINDOW;
- il colore da usare per la finestra dopo la cancellazione.

Essa ritorna un valore numerico di stato che, quando è uguale a zero, indica che l'operazione si è conclusa in maniera positiva.

```
FUNCTION CLEARWINDOW (OpWin, Col)
        IF OpWin <1 OR OpWin >MAXW THEN
            CLEARWINDOW = -4            ' Invalid Window Number
            EXIT FUNCTION
        END IF
        IF Windows(OpWin).Used = 0 THEN
            CLEARWINDOW = -5            ' No Window Open
            EXIT FUNCTION
        END IF
        PL = 0
        IF Windows(OpWin).Crn > 0 THEN
            PL = 1
        END IF
        Xt = Windows(OpWin).InCol + PL -1
```

```
        Yt = Windows(OpWin).InRow + PL -1
        Xb = Windows(OpWin).InCol + Windows(OpWin).Cols - PL - 1
        Yb = Windows(OpWin).InRow + Windows(OpWin).Rows - PL - 1
        CLEARBOX Xt, Yt, Xb-1, Yb-1, Col
        CLEARWINDOW = 0
END FUNCTION
```

Nome : **CLOSEPOPMENU**

È la funzione complementare alla OPENPOPMENU; essa, infatti, chiude un menu aperto con la suddetta funzione basandosi sul numero di Menu (primo parametro) e sulle informazioni conservate in precedenza dall OPENPOPMENU nella variabile Handle$ (secondo parametro). La CLOSEPOPMENU usa la funzione RESTBOX che permette di deallocare la memoria occupata in precedenza e visualizzare nell'area occupata dal menu, quanto esisteva prima dell'apertura dello stesso.

```
SUB CLOSEPOPMENU (Menu, Handle$)
        IF Handle$ = "" THEN
                EXIT SUB
        END IF
        Coo1 = INSTR(Handle$, "*")
        Xtop = VAL(MID$(Handle$, Coo1+1))
        Coo2 = INSTR(Coo1+1, Handle$, "/")
        Ytop = VAL(MID$(Handle$, Coo2+1))
        Coo3 = INSTR(Coo2+1, Handle$, "/")
        Xbot  = VAL(MID$(Handle$, Coo3+1))
        Coo4 = INSTR(Coo3+1, Handle$, "/")
        Ybot  = VAL(MID$(Handle$, Coo4+1))
        RESTBOX Xtop, Ytop, Xbot, Ybot, Handle$
        PosLenTitPri = INSTR(Coo4+1, Handle$, "*")
        LenTitPri = VAL(MID$(Handle$, PosLenTitPri+1))
        LOCATE 1, Xtop
        PRINT " ";
        LOCATE 1, Xtop+LenTitPrI+1
        PRINT " ";
        COLOR 15,0
END SUB
```

Nome : **CLOSEWINDOW**

È la funzione opposta della OPENWINDOW. Essa serve a chiudere una finestra aperta in precedenza ed a rimettere a posto il testo che era scritto sotto quest'ultima. Viene liberata la memoria occupata per tale finestra e viene restituito un codice d'errore che, se uguale a zero, indica che l'operazione è stata conclusa con successo. Essa ammette un solo parametro e cioè il numero della finestra da chiudere, numero fornito in precedenza dall OPENWINDOW.

```
FUNCTION CLOSEWINDOW (OpWin)
        IF OpWin<1 OR OpWin>MAXW THEN
                CLOSEWINDOW = -4            ' Invalid Window Number
                EXIT FUNCTION
```

```
        END IF
        IF Windows(OpWin).Used = 0 THEN
                CLOSEWINDOW = -5            ' No Window Open
                EXIT FUNCTION
        END IF
        Xt = Windows(OpWin).InCol
        Yt = Windows(OpWin).InRow
        Xb = Xt + Windows(OpWin).Cols - 1
        Yb = Yt + Windows(OpWin).Rows - 1
        Hdl$ = STR$(Windows(OpWin).PrecAreaSeg) + ":0"
        RESTBOX Xt, Yt, Xb, Yb, Hdl$
        Windows(OpWin).Used = 0
        CLOSEWINDOW = 0                     ' Ok
END FUNCTION
```

Nome : **CMPDATE**

È una funzione che restituisce un valore intero e serve a confrontare due date, fornite come parametri nel formato **gg/mm/aaaa**. Il risultato è codificato nella maniera seguente

Valore restituito	Risultato
1	La prima data è maggiore della seconda
0	Le due date sono eguali
-1	La prima data è minore della seconda

```
DEFINT A-Z
FUNCTION CMPDATE (D1$, D2$)
        DA$ = RIGHT$(D1$, 4)+MID$(D1$,4,2)+LEFT$(D1$,2)
        DB$ = RIGHT$(D2$, 4)+MID$(D2$,4,2)+LEFT$(D2$,2)
        IF DA$>DB$ THEN
                CMPDATE=1
        ELSEIF DA$=DB$ THEN
                CMPDATE=0
        ELSE
                CMPDATE=-1
        END IF
END FUNCTION
```

Nome : **COLORWINDOW**

È una funzione che permette di cambiare il colore, sia di primo piano che di sfondo, di una finestra già aperta. È importante che questa non sia sovrascritta da un'altra in quanto il nuovo colore è posto direttamente in memoria video. Essa ha due parametri che sono

- il numero della finestra ritornato dalla OPENWINDOW;
- il codice numerico del nuovo colore da attribuire.

Come per le altre funzioni di gestione delle finestre, essa ritorna un valore numerico che quando è diverso da zero, indica una condizione di errore.

```
DEFINT A-Z
FUNCTION COLORWINDOW (OpWin, Col)
	IF OpWin<1 OR OpWin>MAXW THEN
		COLORWINDOW = -4		‘ Invalid Window Number
		EXIT FUNCTION
	END IF
	IF Windows(OpWin).Used = 0 THEN
		COLORWINDOW = -5		‘ No Window Open
		EXIT FUNCTION
	END IF
	VSEG=GETVIDEOSEG
	VOff=(Windows(OpWin).InRow-1)*160+(Windows(OpWin).InCol-1)*2
	DEF SEG=VSEG
	IVOFF=VOff+1
	NR=Windows(OpWin).Rows
	DO WHILE NR > 0
		NC = Windows(OpWin).Cols
		DO WHILE NC > 0
			POKE IVOFF,Col
			IVOFF=IVOFF+2
			NC=NC-1
		LOOP
		IVOFF=IVOFF+160-2*Windows(OpWin).Cols
		NR=NR-1
	LOOP
	DEF SEG
	COLORWINDOW=0			‘ OK
END FUNCTION
```

Nome : **CONVATTR$**

È una funzione che ritorna una stringa contenente i codici letterali degli attributi di un file, partendo dal suo parametro numerico intero in cui questi sono codificati secondo la seguente tabella

1	**=**	**attributo R**
2	**=**	**attributo H**
4	**=**	**attributo S**
8	**=**	**attributo V**
16	**=**	**attributo D**
32	**=**	**attributo A**

Nel caso in cui il codice non corrisponda ad alcun attributo, viene ritornato il carattere N (Not closed file).

```
DEFSNG A-Z
FUNCTION CONVATTR$(CATTR AS INTEGER)
	FA$=””
```

```
        IF (CATTR AND 1) = 1 THEN FA$=FA$ + "R"
        IF (CATTR AND 2) = 2 THEN FA$=FA$ + "H"
        IF (CATTR AND 4) = 4 THEN FA$=FA$ + "S"
        IF (CATTR AND 8) = 8 THEN FA$=FA$ + "V"
        IF (CATTR AND &H10) = &H10 THEN FA$=FA$ + "D"
        IF (CATTR AND &H20) = &H20 THEN FA$=FA$ + "A"
        IF FA$="" THEN
                FA$="N"
        END IF
        CONVATTR$=FA$
END FUNCTION
```

Nome : **CURRDISK$**

Questa funzione alfanumerica restituisce una stringa contenente il disco correntemente in uso dal S.O. Essa non ha alcun parametro ed il dato restituito è nel formato richiesto dalle funzioni CURRPATH$, DISKFREE&, DISKTYPE$, GETLABEL$ e SETDISK.

```
FUNCTION CURRDISK$
        Reg.AX = &H1900
        INTERRUPT &H21, Reg, Reg
        CURRDISK$ = CHR$((Reg.AX AND &HFF) + 65) + ":"
END FUNCTION
```

Nome : **CURRPATH$**

È una funzione alfanumerica che restituisce una stringa contenente il path corrente del disco indicato come parametro. Per tale indicazione è sufficiente una stringa in cui il primo carattere è, in maiuscolo, la lettera del drive di cui si vuole l'informazione. Nella stringa ritornata, viene incluso, all'inizio, il nome del drive ed alla fine un carattere ASCII zero che può essere eliminato con la funzione LEFT$.

```
FUNCTION CURRPATH$
        CPH$ = SPACE$(64)
        Regx.AX = &H4700
        Regx.DX = ASC(UCASE$(DISK$)) - 64
        Regx.SI = SADD(CPH$)
        Regx.DS = VARSEG(CPH$)
        INTERRUPTX &H21, Regx, Regx
        CURRPATH$ = RTRIM$(UCASE$(DISK$) + "\" +  CPH$)
END FUNCTION
```

Nome : **CURSORWINDOW**

È una delle funzioni di gestione delle finestre testo. Essa consente di spostare il cursore all'interno di una finestra per potere visualizzare dati in punti diversi della stessa. Le coordinate sono relative alla posizione della finestra e costituiscono il secondo e terzo parametro che bisogna fornire (colonna e riga rispettivamente). Il primo parametro è, invece, il numero della finestra utilizzata e che è stato fornito dalla funzione OPENWINDOW all'apertura della stessa.
La funzione ritorna un valore intero che, se diverso da zero, indica una condizione di errore verificatasi durante l'esecuzione della stessa.

```
DEFINT A-Z
FUNCTION CURSORWINDOW (OpWin, Row, Col)
	IF OpWin<1 OR OpWin>MAXW THEN
		CURSORWINDOW = -4			‘ Invalid Window Number
		EXIT FUNCTION
	END IF
	IF Windows(OpWin).Used = 0 THEN
		CURSORWINDOW = -5			‘ No Window Open
		EXIT FUNCTION
	END IF
	IF Windows(OpWin).Crn > 0 THEN
		PL=1
	END IF
	IF Row <1 + PL OR Col <1 + PL THEN
		CURSORWINDOW = -2			‘ Coordinate Error
		EXIT FUNCTION
	END IF
	IF Row > Windows(OpWin).Rows - PL OR Col > Windows(OpWin).Cols - PL THEN
		CURSORWINDOW = -2			‘ Coordinate Error
		EXIT FUNCTION
	END IF
	Windows(OpWin).CurRow = Row
	Windows(OpWin).CurCol = Col
	CURSORWINDOW = 0				‘ Ok
END FUNCTION
```

Nome : **DISKFREE&**

Questa funzione provvede a ritornare un valore intero lungo che rappresenta il numero dei bytes ancora liberi sul disco contenuto nel drive indicato come parametro (prima lettera del parametro in maiuscolo). La funzione è valida sia per i floppy-disk (drive A: e B:) che per gli hard-disk (drive C:, D: e seguenti).

```
DEFSNG A-Z
FUNCTION DISKFREE& (DISK$)
	Reg.AX = &H3600
	Reg.DX = ASC(UCASE$(DISK$)) - 64
	INTERRUPT &H21, Reg, Reg
	DISKFREE& = CLNG(Reg.AX) * Reg.BX * Reg.CX
END FUNCTION
```

Nome : **DISKTYPE$**

Con questa funzione, di tipo alfanumerico, è possibile conoscere il tipo di disco inserito in un determinato drive. Essa ha come unico parametro una stringa indicante, in maiuscolo con una lettera, il drive in cui è inserito il disco in test (A: e B: per i floppy disk, C:, D: e seguenti per gli hard disk). La stringa ritornata può essere una tra le seguenti

5” ¼ DD

5” ¼ DD (8 sec)
5” ¼ HD
3” ½ DD
3” ½ HD
Hard Disk
SCONOSCIUTO

```
FUNCTION DISKTYPE$(DISK$)
        Regx.AX = &H1C00
        Regx.DX = ASC(UCASE$(DISK$)) - 64
        INTERRUPTX &H21, Regx, Regx
        DEF SEG = Regx.DS
        DT% = PEEK(Regx.BX) - &HEF
        DEF SEG
        SELECT CASE DT%
                CASE 1
                        DT$ = “3” + CHR$(34) + “ + HD”
                CASE 10
                        IF (Regx.AX AND &HFF) = 1 THEN
                                DT$ = “5” + CHR$(34) + “, HD”
                        ELSE
                                DT$ = “3” + CHR$(34) + “ + DD”
                        END IF
                CASE 14, 16
                        DT$ = “5” + CHR$(34) + “, DD”
                        IF DT% = 16 THEN
                                DT$ = DT$ + “ (8 sec)”
                        END IF
                CASE 9
                        DT$ = “Hard Disk”
                CASE ELSE
                        DT$ = “SCONOSCIUTO”
        END SELECT
        DISKTYPE$ = DT$
END FUNCTION
```

Nome : **DOSVER$**

È una semplice funzione alfanumerica che ritorna una stringa contenente il numero di versione del sistema operativo. È utile quando si devono eseguire comandi e funzioni di S.O. presenti a partire da una determinata versione in poi. La funzione non ammette alcun parametro in ingresso.

```
FUNCTION DOSVER$
        Reg.AX = &H3000
        INTERRUPT &H21, Reg, Reg
        DMJ$ = LTRIM$(STR$(Reg.AX AND &HFF))
        DMN$ = LTRIM$(STR$((Reg.AX AND &HFF00) / &H100))
        DOSVER$=DMJ$ + “.” + DMN$
```

```
END FUNCTION
```

Nome : **DWEEK**

È una funzione che ritorna un valore intero compreso tra 0 e 6 corrispondente al giorno della settimana di una data inserita come parametro in ingresso (0=Domenica, 1=Lunedì, 2=Martedì, …). Tale parametro, alfanumerico, è nel formato **gg/mm/aaaa**. È possibile sfruttare l'array globale predefinito di nome **Giorno$** per ottenere, usando come indice il risultato di tale funzione, una stringa con il nome del giorno della settimana richiesto; per fare ciò, bisogna includere nel sorgente il file DATETIME.INC. Se la data in ingresso non dovesse essere corretta, verrebbe ritornato dalla funzione il valore -1. Le date valide sono comprese tra il 01/01/1980 ed il 31/12/2099.

```
DEFINT A-Z
FUNCTION DWEEK (DT$)
	G=VAL(DT$)
	M=VAL(MID$(DT$, 4 , 2))
	A=VAL(RIGHT$(DT$, 4))
	D$=DATE$
	Reg.CX = A
	Reg.DX = M * &H100& + G
	Reg.AX = &H2B00
	INTERRUPT &H21, Reg, Reg
	IF (Reg.AX AND &HFF) = 0 THEN
		Reg.AX = &H2A00
		INTERRUPT &H21, Reg, Reg
		DWEEK = Reg.AX AND &HFF
	ELSE
		DWEEK = -1		' Error
	END IF
	DATE$ = D$
END FUNCTION
```

Nome : **ETA**

Funzione numerica che accetta due parametri alfanumerici, ambedue nel formato **gg/mm/aaaa**, e che ritorna un valore intero che rappresenta il numero di anni passati dalla prima alla seconda data; se, come secondo parametro, viene passata la data odierna, si può calcolare l'età di un individuo precisando la sua data di nascita nel primo parametro. In caso di date incoerenti (la prima maggiore della seconda), viene ritornato un valore d'errore (-1)

```
DEFINT A-Z
FUNCTION ETA(DN$, DO$)
	IF CMPDATE(DN$, DO$) = 1 THEN
		ETA=-1
		EXIT FUNCTION
	END IF
	AN = VAL(RIGHT$(DN$, 4))
	AO = VAL(RIGHT$(DO$, 4))
	ET = AO - AN
```

```
	MN = VAL(MID$(DN$, 4, 2))
	MO = VAL(MID$(DO$, 4, 2))
	IF MN>MO THEN
		ETA = ET-1
		EXIT FUNCTION
	ELSE
		GN = VAL(LEFT$(DN$, 2))
		OG = VAL(LEFT$(DO$, 2))
		IF GN>OG THEN
			ETA = ET-1
			EXIT FUNCTION
		END IF
	END IF
	ETA = ET
END FUNCTION
```

Nome : **FILEDATE$**

È una funzione usata per lo più da altre routines della libreria, che restituisce una stringa nel formato **gg/mm/aaaa**, dopo aver convertito il suo unico parametro alfanumerico in cui viene posta una data, nel formato usato da MS-DOS all'interno delle directory per ogni file

```
DEFINT A-Z
FUNCTION FILEDATE$(D$) STATIC
	D = CVI(D$)
	G = D AND &H1F
	M = (D AND &H1E0) / &H20
	A = (D AND &HFE00) / &H200 + 1980
	FD$ = ADDZERO2STR$(CSNG(G), 2) + "/"
	FD$ = FD$ + ADDZERO2STR$(CSNG(M), 2) + "/"
	FD$ = FD$ + ADDZERO2STR$(CSNG(A), 4)
	FILEDATE$ = FD$
END FUNCTION
```

Nome : **FILEEXIST**

È una funzione che ritorna un valore numerico intero, in base all'esistenza di un determinato file usato come parametro. È possibile specificare sia il drive che il path completo e, naturalmente, non sono accettati caratteri jolly ed il nome del file deve essere, eventualmente, specificato completo di estensione. I valori ritornati sono

0 se il file specificato non esiste
-1 se il file specificato esiste (ha lunghezza maggiore di 0)

```
DEFINT A-Z
FUNCTION FILEEXIST(FILE$)
	NF = FREEFILE
	OPEN FILE$ FOR BINARY AS NF
	LN& = LOF(NF)
	CLOSE NF
```

```
	IF LN& > 0 THEN
		FILEEXIST = -1
	ELSE
		KILL FILE$
		FILEEXIST = 0
	END IF
END FUNCTION
```

Nome : **FILETIME$**

Come per la FILEDATE$, la FILETIME$ è una funzione usata da alcune routines interne alla libreria. Essa serve a convertire nel formato **hh:mm:ss** il parametro alfanumerico ad essa fornito; quest'ultimo è nel formato dell'orario usato per i file nella directory da MS-DOS

```
DEFINT A-Z
FUNCTION FILETIME$ (T$) STATIC
	T = CVI(T$)
	S = (T AND &H1F) * 2
	M = (T AND &H7E0) / &H20
	H = (T AND &HF800) / &H800
	IF H<0 THEN
		H=ABS(H)+12
	END IF
	FT$ = ADDZERO2STR$(CSNG(H), 2) + “:”
	FT$ = FT$ + ADDZERO2STR$(CSNG(M), 2) + “:”
	FT$ = FT$ + ADDZERO2STR$(CSNG(S), 2)
	FILETIME$ = FT$
END FUNCTION
```

Nome : **GETDCC$**

Tale funzione alfanumerica restituisce una stringa il cui contenuto indica la combinazione di scheda graficae monitor disponibili nel sistema in uso. Essa non ha alcun parametro e la stringa restituita può essere una tra le seguenti

No Display
MDA
CGA
EGA Color
EGA Mono
PGA
VGA Mono
VGA Color
MCGA Mono
MCGA Color
Sconosciuto

```
FUNCTION GETDCC$
	Reg.AX = &H1A00
	INTERRUPT &H10, Reg, Reg
```

```
	SELECT CASE (Reg.BX AND &HFF)
		CASE 0
			RS$ = "No Display"
		CASE 1
			RS$ = "MDA"
		CASE 2
			RS$ = "CGA"
		CASE 4
			RS$ = "EGA Color"
		CASE 5
			RS$ = "EGA Mono"
		CASE 6
			RS$ = "PGA"
		CASE 7
			RS$ = "VGA Mono"
		CASE 8
			RS$ = "VGA Color"
		CASE &HB
			RS$ = "MCGA Mono"
		CASE &HC
			RS$ = "MCGA Color"
		CASE ELSE
			RS$ = "Sconosciuto"
	END SELECT
	GETDCC$ = RS$ + "."
END FUNCTION
```

Nome : **GETDTA**

La routine GETDTA restituisce nelle due variabili globali di nome **DtaSeg** e **DtaOff**, rispettivamente, il segmento e l'offset della DTA (Disk Transfer Area) corrente. Tale informazione è utile se si usano alcune funzioni del sistema operativo per il trattamento diretto (a livello di tracce e settori) delle informazioni contenute su disco. Essa infatti, costituisce il buffer attraverso il quale passano le informazioni lette o scritte dal S.O. sul disco

```
SUB GETDTA
	Regx.AX = &H2F00
	INTERRUPTX &H21, Regx, Regx
	DtaSeg = Regx.ES
	DtaOff = Regx.BX
END SUB
```

Nome : **GETFILEATTR$**

Questa funzione accetta, come primo parametro, una stringa contenente un nome di file completo, eventualmente, di drive e path, e ritorna una stringa contenente gli attributi attuali dello stesso

```
FUNCTION GETFILEATTR$(FILE$)
	F$ = FILE$ + CHR$(0)
```

```
        Regx.AX = &H4300
        Regx.DX = SADD(F$)
        Regx.DS = VARSEG(F$)
        INTERRUPTX &H21, Regx, Regx
        IF (Regx.FLAGS AND 1) = 1 THEN
                GETFILEATTR$ = "?"
        ELSE
                GETFILEATTR$ = CONVATTR$(Regx.CX)
        END IF
END FUNCTION
```

Nome : **GETKEYMENU**

Questa è una delle funzioni per la gestione dei popup menu. Essa provvede a gestire un singolo menu tra quelli abilitati, con un massimo di 20 scelte. È usata dalla SELECTOPZMENU! nella gestione di più popup menus; essa accetta un solo parametro numerico, il numero del menu da gestire. Fa uso della OPENPOPMENU e della CLOSEPOPMENU ed, inoltre, di tutti gli arrays definiti nel file include POPMENU.INC.
La funzione ritorna un numero intero secondo la seguente tabella

0 è stato premuto ESC
-2 è stato premuto il tasto per lo spostamento del cursore a destra
-4 è stato premuto il tasto per lo spostamento del cursore a sinistra

Se viene premuto il tasto Return, viene ritornato dalla funzione il numero d'ordine dell'opzione all'interno del menu

```
FUNCTION GETKEYMENU(Menu)
        DIM Opz$(1 TO 20)
        El = 0
        ActPos = INSTR$(PopMenus$(Menu), "*") + 1
        DO WHILE ActPos < LEN(PopMenus$(Menu))
                NewPos = INSTR(ActPos, PopMenus$(Menu), "*") + 1
                EndOpz = NewPos – ActPos – 1
                El = El + 1
                Opz$(El) = MID$(PopMenus$(Menu), ActPos, EndOpz)
                ActPos = NewPos
        LOOP
        Info = OPENPOPMENU (Menu, H$)
        PosPri = Info AND 255
        MaxX = Info \ 256
        Opz = 1
        AKey$ = ""
        DO WHILE AKey$<>CHR$(13) AND AKey$<>CHR$(27)
                COLOR 15, 0
                LOCATE 2+Opz, PosPri
                PRINT " "; Opz$(Opz); SPACE$(MaxX + 1 - LEN(Opz$(Opz)));
                AKey$=""
                DO WHILE LEN(AKey$)=0
```

```
			AKey$=INKEY$
		LOOP
		COLOR 0, 15
		KCode = ASC(AKey$)
		IF LEN(AKey$)>1 THEN
			KCode=ASC(MID$(AKey$, 2))
			IF KCode=72 OR KCode=80 THEN
				COLOR 0, 15
				LOCATE 2+Opz, PosPri
				PRINT " "; Opz$(Opz); SPACE$(MaxX + 1 - LEN(Opz$(Opz)));
			END IF
			SELECT CASE KCode
				CASE 72
					Opz = Opz - 1
					IF Opz = 0 THEN
						Opz = El
					END IF
				CASE 80
					Opz = Opz + 1
					IF Opz > El THEN
						Opz = 1
					END IF
				CASE 75, 77
					EXIT DO
				CASE ELSE
			END SELECT
		END IF
	LOOP
	CLOSEPOPMENU Menu, H$
	SELECT CASE KCode
		CASE 13
			GETKEYMENU = Opz
		CASE 27
			GETKEYMENU = 0
		CASE 75, 77
			GETKEYMENU = KCode - 79
	END SELECT
END FUNCTION
```

Nome : **GETLABEL$**

Questa funzione ha come parametro una stringa contenente, in maiuscolo, la lettera di un drive; essa ritorna un'altra stringa contenente il nome del disco in esso contenuto, se assegnato, altrimenti la frase **No Label**. Il drive specificato deve essere, naturalmente, valido

```
FUNCTION GETLABEL$(DISK$)
	SEARCH$ = DISK$ + "\*.*" + CHR$(0)
	GETDTA
	SETDTA VARSEG(Dir), VARPTR(Dir)
```

```
        Regx.AX = &H4E00
        Regx.CX = 8
        Regx.DX = SADD(SEARCH$)
        Regx.DS = VARSEG(SEARCH$)
        INTERRUPTX &H21, Regx, Regx
        IF (Regx.FLAGS AND 1) = 1 AND (Regx.AX AND &HFF) = &H12 THEN
                GETLABEL$ = "No Label"
        ELSE
                P=INSTR(Dir.FILENAME, ".")
                IF P THEN
                   GETLABEL$ = RTRIM$(LEFT$(Dir.FILENAME, P-1)+MID$(Dir.FILENAME, P+1))
                ELSE
                   GETLABEL$ = RTRIM$(Dir.FILENAME)
                END IF
        END IF
        SETDTA DtaSeg, DtaOff
END FUNCTION
```

Nome : **GETMOUSEBUTTON**

È una funzione di gestione del mouse che ritorna un valore intero indicante lo stato dei tasti del mouse stesso. Come per tutte le funzioni di gestione del mouse, prima di essere usata, deve essere stato attivato il driver software che lo controlla

```
FUNCTION GETMOUSEBUTTON
        DUM = MOUSE(3, 0, 0, BTS)
        GETMOUSEBUTTON = BTS
END FUNCTION
```

Nome : **GETMOUSEPOS**

È una routine che accetta, come parametri, due variabili intere il cui contenuto iniziale non ha importanza. All'uscita esse contengono, rispettivamente, la colonna e la riga in cui è posizionato attualmente il mouse. Come per le altre routines di gestione mouse, il driver software dello stesso deve essere stato attivato in precedenza

```
SUB GETMOUSEPOS (POSX, POSY)
        DUM = MOUSE(3, POSX, POSY, 0)
END SUB
```

Nome : **GETVIDEOSEG**

È una funzione che ritorna un valore intero indicante il segmento attuale della memoria video. Essa non usa alcun parametro in ingresso

```
FUNCTION GETVIDEOSEG
        Reg.AX = &HF00
        INTERRUPT &H10, Reg, Reg
        VS = &HA000
        SELECT CASE (Reg.AX AND &HFF)
                CASE 0 TO 6
```

```
			VS=&HB800
		CASE 7 TO 10
			VS=&HB000
		CASE ELSE
			VS=&HA000
	END SELECT
	GETVIDEOSEG = VS
END FUNCTION
```

Nome : **INVERSE**

Questa funzione permette di visualizzare una riga all'interno di una finestra con i colori in reverse al fine di evidenziarla. Essa viene usata principalmente dalla funzione MCHOICE per evidenziare i dati all'interno della finestra apposita. I parametri sono 4 e precisamente

OpWin Handle della finestra considerata
Inv Punto di inizio della zona da evidenziare
Largh Larghezza della riga da evidenziare in caratteri
Col Colore della riga; se negativo non viene evidenziata la riga

```
SUB INVERSE (OpWin, Inv, Largh, Col)
	DEF SEG = GETVIDEOSEG
	VOff = (Windows(OpWin).InRow-1+Inv) * 160+(Windows(OpWin).InCol)*2
	IF Col<0 THEN
		PokeCol = -Col
	ELSE
		PokeCol = Col XOR 32
	END IF
	FOR LFr=1 TO Largh
		POKE VOff+1, PokeCol
		VOff=VOff+2
	NEXT LFr
	DEF SEG
END SUB
```

Nome : **ISDIGIT**

È una funzione di tipo logico che ritorna un valore intero a seconda del contenuto del suo parametro alfanumerico. Se quest'ultimo, costituito da un carattere ASCII, è uno dei seguenti

. - + 0 1 2 3 4 5 6 7 8 9 0

la funzione ritorna il valore -1, altrimenti ritorna il valore 0

```
FUNCTION ISDIGIT(Ch)
	SELECT CASE Ch
		CASE 43, 45, 46
			ISDIGIT = -1
		CASE IS < 48
			ISDIGIT = 0
```

```
        CASE IS > 57
            ISDIGIT = 0
        CASE ELSE
            ISDIGIT = -1
    END SELECT
END FUNCTION
```

Nome : **MCHOICE**

Questa funzione permette di gestire un elenco di dati in maniera semplificata. Tali dati, caricati nel vettore alfanumerico **ArData$()**, vengono visualizzati all'interno di una finestra le cui dimensioni sono definite dai primi parametri (**Xt, Yt, Xb, Yb**); è possibile, inoltre, definire una stringa (**Title$**) che verrà posta in cima alla finestra ed un'altra stringa (**Bottom$**) alla fine della stessa. Il parametro **Col** è il codice del colore combinato (background/foreground) della finestra.
Infine, il parametro **SelPos**, quando diverso da zero, indica la colonna della finestra in cui dovrà apparire un segno se viene premuto lo spazio durante l'uso della funzione.
Il valore ritornato dalla funzione indica il numero della riga ultima selezionata

```
DEFINT A-Z
FUNCTION MCHOICE(Xt, Yt, Xb, Yb, Title$, Bottom$, Col, ArData$(), SelPos)
    WMCH=OPENWINDOW(Xt, Yt, Xb, Yb, Col, 2)
    Lungh=Yb-Yt-1
    Largh=Xb-Xt-1
    COLOR 15, Col \ 16
    IF Title$<>"" THEN
        LOCATE Yt, Xt+1
        PRINT UCASE$(LEFT$(" " + Title$ + " ", Xb-Xt));
    END IF
    IF Bottom$<>"" THEN
        Btm$=LEFT$(" " + Bottom$ + " ", Xb-Xt)
        LOCATE Yb, (Largh-LEN(Btm$)) / 2 + Xt
        PRINT Btm$;
    END IF
    Act = 1
    Inv = 1
    DO
        AKey$=""
        Ps=1
        DO WHILE Ps<=Lungh
            ST=CURSORWINDOW(WMCH, Ps+1, 2)
            IF Ps>UBOUND(ArData$) THEN
                DToVis$=SPACE$(Largh-1)
            ELSE
                DToVis$=LEFT$(ArData$(Act+Ps-1), Largh)
            END IF
            ST=PRINTWINDOW(WMCH, DToVis$, Col, 0)
            Ps = Ps + 1
        LOOP
```

```
INVERSE WMCH, Inv, Largh, Col
DO WHILE AKey$=""
	AKey$=INKEY$
	LK=LEN(AKey$)
	IF LK>0 THEN
		INVERSE WMCH, Inv, Largh, Col
	END IF
	IF LK=1 THEN
		AKey=ASC(AKey$)
		SELECT CASE AKey
			CASE 13
				MCHOICE = Inv+Act-1
				ST=CLOSEWINDOW(WMCH)
				EXIT FUNCTION
			CASE 27
				MCHOICE = 0
				ST=CLOSEWINDOW(WMCH)
				EXIT FUNCTION
			CASE 32
				IF SelPos > 0 THEN
				   SelCh$ = " "
				   IF MID$(ArData$(Inv+Act-1), SelPos, 1)=" " THEN
				      SelCh$=CHR$(251)
				   END IF
				   MID$(ArData$(Inv+Act-1), SelPos, 1)=SelCh$
				END IF
			CASE ELSE
		END SELECT
	END IF
	IF LK=2 THEN
		AKey=ASC(MID$(AKey$, 2))
		SELECT CASE AKey
			CASE 72
				Inv=Inv-1
				IF Inv=0 THEN
					IF Act=1 THEN
						BEEP
						Inv=Inv+1
					ELSE
						Act=Act-1
						Inv=Inv+1
					END IF
				END IF
			CASE 80
				Inv=Inv+1
				IF Inv>UBOUND(ArData$) OR Inv>Lungh THEN
					IF Act+Lungh>UBOUND(ArData$) THEN
						Inv=Inv-1
```

```
                        BEEP
                    ELSE
                        Act=Act+1
                        Inv=Inv-1
                    END IF
                END IF
            CASE 73
                Act=Act-Lungh/2
                IF Act<1 THEN
                    Act=1
                END IF
            CASE 81
                Act=Act+Lungh/2
                Limit=UBOUND(ArData$)-Lungh+1
                IF Act>Limit THEN
                    Act=Limit
                END IF
                IF Limit<0 THEN
                    Act=1
                END IF
            CASE 132
                Inv=1
                Act=1
            CASE 118
                Act=UBOUND(ArData$)-Lungh+1
                Inv=Lungh
                IF Act<0 THEN
                    Act=1
                    Inv=UBOUND(ArData$)
                END IF
            CASE 71
                Inv=1
            CASE 79
                Inv=Lungh
                IF Lungh>UBOUND(ArData$) THEN
                    Inv=UBOUND(ArData$)
                END IF
            CASE ELSE
        END SELECT
    END IF
  LOOP
 LOOP
END FUNCTION
```

Nome : **MOUSE**

È la funzione generica di interfaccia con il driver software di controllo del mouse. Essa ammette 4 parametri, che sono, nell'ordine

- un valore intero indicante l'operazione richiesta al driver software;

- le due coordinate, colonna e riga, iniziali del mouse;

- una variabile intera in cui sarà restituito lo stato dei pulsanti del mouse.

La funzione restituisce l'esito dell'operazione richiesta e, se si usano due variabili intere per le coordinate, le coordinate attuali del cursore del mouse. Ovviamente, quanto precisato nelle altre routines di gestione del mouse per quanto riguarda il driver software dello stesso, vale anche per questa funzione

```
FUNCTION MOUSE(OPER, COOX, COOY, BUTTON)
        Reg.AX=OPER
        Reg.CX=COOX
        Reg.DX=COOY
        INTERRUPT &H33, Reg, Reg
        COOX=Reg.CX
        COOY=Reg.DX
        BUTTON=Reg.BX
        MOUSE=Reg.AX
END FUNCTION
```

Nome : **MOUSECHECK**

È una funzione di controllo che ritorna il valore -1 se il driver software del mouse è attivato, altrimenti il valore 0. È usata per controllare, all'inizio di ogni programma applicativo che usi il mouse, che lo stesso possa essere sfruttato

```
FUNCTION MOUSECHECK
        MOUSECHECK=MOUSE(0, 0, 0, 0)
END FUNCTION
```

Nome : **MOUSEOFF**

Questa routine, che non usa alcun parametro, serve solamente a disattivare il cursore del mouse sul video. Anche in questo caso il driver software del mouse deve essere stato attivato in precedenza

```
SUB MOUSEOFF
        DUM = MOUSE(2, 0, 0, 0)
END SUB
```

Nome : **MOUSEON**

La routine MOUSEON, al contrario della MOUSEOFF, se è stato attivato il driver software del mouse, attiva il cursore dello stesso sul video

```
SUB MOUSEON
	DUM = MOUSE(1, 0, 0, 0)
END SUB
```

Nome : **OPENPOPMENU**

È una delle due funzioni di base per la gestione dei popup menu insieme alla CLOSEPOPMENU. Essa consente di aprire una finestra di tipo testo e visualizzare all'interno di questa tutte le opzioni definite in precedenza. La OPENPOPMENU usa la funzione SAVEBOX che provvede ad allocare la memoria necessaria ed a preservare il precedente contenuto dell'area del video interessata. Il primo parametro (**Menu**) rappresenta il numero del menu da gestire tra tutti quelli previsti; nel secondo (variabile **Handle$**) la funzione ritorna delle informazioni necessarie alla CLOSEPOPMENU per chiudere il menu aperto. La OPENPOPMENU inoltre, ritorna, all'uscita, un numero intero in cui, codificate, vengono fornite delle indicazioni riguardanti la colonna di inizio e fine del menu attivato

```
FUNCTION OPENPOPMENU(Menu, Handle$)
	COLOR 0, 15
	POSX = 3
	Tit = 1
	DO WHILE Tit<Menu
		LenTStr=INSTR(PopMenu$(Tit), "*") - 1
		POSX=POSX + LenTStr + 2
		Tit=Tit+1
	LOOP
	Els=1
	AsPoint=INSTR(PopMenu$(Tit), "*") + 1
	DO WHILE AsPoint<LEN(PopMenu$(Menu))
		AsPoint=INSTR(AsPoint, PopMenu$(Menu), "*") + 1
		IF AsPoint > 0 THEN
			Els=Els+1
		END IF
	LOOP
	DIM ActMenu$(1 TO Els)
	MaxX=0
	AsStart=INSTR(PopMenu$(Tit), "*") + 1
	AsPoint=AsStart
	LenTitPri=AsStart-2
	El=1
	DO WHILE AsPoint<LEN(PopMenu$(Menu))
		AsPoint=INSTR(AsPoint, PopMenu$(Menu), "*") + 1
		IF AsPoint>0 THEN
			ActMenu$(El)=MID$(PopMenu$(Menu), AsStart, AsPoint-AsStart-1)
			AsStart=AsPoint
			IF LEN(ActMenu$(El)) > MaxX THEN
```

```
                        MaxX=LEN(ActMenu$(El))
                    END IF
                    El = El + 1
            END IF
    LOOP
    Handle$=SAVEBOX$(POSX-1, 2, POSX+MaxX+2, 2+El)
    IF Handle$=”*” THEN
            EXIT FUNCTION
    END IF
    Handle$=Handle$ + “*” + STR$(LenTitPri)
    LOCATE 1, POSX-1
    PRINT “|”;
    LOCATE 1, POSX+LenTitPri
    PRINT “|”;
    LOCATE 2, POSX-1
    PRINT “+”; STRING$(MaxX+2, 196); “+”
    LOCATE 2, POSX+LenTitPri
    IF LenTitPri=MaxX+2 THEN
            PRINT “|”;
    ELSE
            PRINT “-“;
    END IF
    FOR Tit2=1 TO El-1
            LOCATE Tit2+2, POSX-1
            PRINT “| ”; ActMenu$(Tit2);SPACE$(MaxX+1-LEN(ActMenu$(Tit2)));
            LOCATE , POSX+MaxX+2
            PRINT “|”;
    NEXT Tit2
    PRINT
    LOCATE , POSX-1
    PRINT “+”; STRING$(MaxX+2, 196); “+”
    OPENPOPMENU=POSX+256*MaxX
END FUNCTION
```

Nome : **OPENWINDOW**

È la funzione che permette di aprire una nuova finestra testo. Essa ammette sei parametri che sono, nell’ordine

- le quattro coordinate degli angoli superiore sinistro ed inferiore destro della finestra (rispettivamente, colonna e riga superiore sinistra e colonna e riga inferiore destra);

- il colore attribuito alla finestra;

- il codice numerico riguardante il tipo di cornice da visualizzare (0=nessuna cornice, 1=cornice semplice, 2=cornice doppia).

La funzione ritorna un numero che, se minore di zero, rappresenta un codice d'errore mentre, quando positivo, è il numero attribuito alla finestra appena aperta; tale numero dovrà essere usato con tutte le funzioni che faranno riferimento a tale finestra.
Il massimo numero di finestre apribili contemporaneamente è fissato dalla costante **MAXW** ed è, attualmente, 10.
I codici d'errore ritornati da questa e da altre funzioni che gestiscono le finestre, sono i seguenti

-1 non è possibile aprire la finestra perché sono state già aperte tutte quelle disponibili;

-2 esiste un errore nelle coordinate specificate per la finestra;

-3 non è disponibile memoria sufficiente per aprire la finestra;

-4 il numero della finestra a cui si fa riferimento non è valido perché fuori dai limiti consentiti;

-5 il numero della finestra a cui si fa riferimento non è valido perché la stessa non è stata mai aperta;

-6 quando la finestra viene chiusa, non è possibile recuperare la memoria occupata in precedenza.

```
FUNCTION OPENWINDOW (Xt, Yt, Xb, Yb, Col, Crn)
	FOR Win=1 TO MAXW
		IF Windows(Win).Used = 0 THEN
			EXIT FOR
		END IF
	NEXT Win
	IF Win>MAXW THEN
		OPENWINDOW=-1		' No Window Available
		EXIT FUNCTION
	END IF
	Largh = Xb-Xt+1
	Lungh = Yb-Yb+1
	Hdl$=SAVEBOX$(Xt, Yt, Xb, Yb)
	IF Hdl$= "*" THEN
		OPENWINDOW = -3		' No Memory Available
		EXIT FUNCTION
	END IF
	Windows(Win).PrecAreaSeg = VAL(Hdl$)
	Windows(Win).InRow = Yt
	Windows(Win).InCol = Xt
	Windows(Win).Rows = Lungh
	Windows(Win).Cols = Largh
	CLEARBOX Xt-1, Yt-1, Xb-1, Yb-1, Col
	VSEG = GETVIDEOSEG
	VOff = (Yt-1)*160+(Xt-1)*2
	IVOFF=VOff
```

```
IAOFF=0
SELECT CASE Crn
	CASE 1
		Cor=196
		Cve=179
		Cas=218
		Cad=191
		Cbs=192
		Cbd=217
	CASE 2
		Cor=205
		Cve=186
		Cas=201
		Cad=187
		Cbs=200
		Cbd=188
	CASE ELSE
END SELECT
IF Crn>0 THEN
	DEF SEG=VSEG
	POKE VOff, Cas
	POKE VOff+1, Col
	POKE VOff + 2*(Largh-1), Cad
	POKE VOff + 2*(Largh-1)+1, Col
	POKE VOff + 160*(Lungh-1), Cbs
	POKE VOff + 160*(Lungh-1)+1, Col
	POKE VOff + 160*(Lungh-1)+2*(Largh-1), Cbd
	POKE VOff + 160*(Lungh-1)+2*(Largh-1)+1, Col
	IVOFF=VOff+2
	NC=Largh-2
	DO WHILE NC>0
		POKE IVOFF, Cor
		POKE IVOFF+1, Col
		POKE IVOFF + (Lungh-1)*160, Cor
		POKE IVOFF + (Lungh-1)*160+1, Col
		IVOFF=IVOFF+2
		NC=NC-1
	LOOP
	IVOFF=VOff+160
	NR=Lungh-2
	DO WHILE NR>0
		POKE IVOFF, Cve
		POKE IVOFF+1, Col
		POKE IVOFF + (Largh-1)*2, Cve
		POKE IVOFF + (Largh-1)*2+1, Col
		IVOFF=IVOFF+160
		NR=NR-1
	LOOP
```

```
            DEF SEG
        END IF
        CR=1
        CC=1
        IF Crn>9 THEN
            CR=2
            CC=2
        END IF
        Windows(Win).CurRow = CR
        Windows(Win).CurCol = CC
        Windows(Win).Crn = Crn
        Windows(Win).Used = -1
        OPENWINDOW = Win
END FUNCTION
```

Nome : **PRINTSCREEN**

Questa routine può essere richiamata per fare l'hard-copy dello schermo sulla stampante, ottenendo lo stesso effetto dell'uso del tasto Print Screen

```
SUB PRINTSCREEN
        INTERRUPT &H5, Reg, Reg
END SUB
```

Nome : **PRINTWINDOW**

Con questa funzione è possibile visualizzare, all'interno di una finestra aperta in precedenza, dei dati alfanumerici rispettandone i margini. Essa ammette 4 parametri di cui, il primo è, naturalmente, il numero attribuito dalla OPENWINDOW alla finestra su cui si vuole operare. Il secondo, invece, è la stringa che si vuole visualizzare, mentre il terzo rappresenta il codice del colore in cui si deve visualizzare quest'ultima. L'ultimo parametro, anch'esso numerico, rappresenta un codice usato per determinare il comportamento della funzione quando i dati da visualizzare vanno oltre una riga, ed ha il seguente significato

-1 la funzione opera in maniera che, alla fine della stringa, il cursore viene portato a capo (in maniera analoga all'istruzione PRINT quando non è seguita da alcun simbolo);

0 alla fine della stringa, il cursore non viene spostato (come per l'istruzione PRINT quando è seguita dal simbolo ';').

La funzione ritorna un codice numerico che è zero quando l'operazione si è conclusa senza errori; esso, invece, è minore di zero quando si è verificato un errore (secondo i codici elencati per la funzione OPENWINDOW)

```
DEFINT A-Z
FUNCTION PRINTWINDOW (OpWin, Astr$, Col, Ret)
        IF OpWin<1 OR OpWin>MAXW THEN
            PRINTWINDOW=-4                  ' Invalid Window Number
            EXIT FUNCTION
        END IF
```

```
IF Windows(OpWin).Used=0 THEN
	PRINTWINDOW=-5			' No Window Open
	EXIT FUNCTION
END IF
IF LEN(Astr$)=0 THEN
	EXIT FUNCTION
END IF
VSEG=GETVIDEOSEG
VOff=(Windows(OpWin).CurRow+Windows(OpWin).InRow-2)*160
VOff= VOff + (Windows(OpWin).CurCol+Windows(OpWin).InCol-2)*2
PL=0
IF Windows(OpWin).Crn>0 THEN
	PL=2
END IF
DEF SEG=VSEG
Idx=1
DO WHILE Idx<=LEN(Astr$)
	POKE VOff, ASC(MID$(Astr$, Idx, 1))
	POKE VOff + 1, Col
	VOff=VOff+2
	Idx=Idx+1
	IF Windows(OpWin).CurCol > Windows(OpWin).Cols-PL THEN
		IF PL=0 THEN
			Windows(OpWin).CurCol=1
		ELSE
			Windows(OpWin).CurCol=2
		END IF
		IF Windows(OpWin).CurRow > Windows(OpWin).Rows-PL THEN
			XST= Windows(OpWin).InCol+(PL/2)-1
			YST= Windows(OpWin).InRow+(PL/2)-1
			XSB= Windows(OpWin).InCol + Windows(OpWin).Cols-PL-1
			YSB= Windows(OpWin).InRow + Windows(OpWin).Rows-PL-1
			SCROLLUPBOX XST, YST, XSB, YSB, 1, Col
		ELSE
			Windows(OpWin).CurRow = Windows(OpWin).CurRow+1
		END IF
		VOff=(Windows(OpWin).CurRow+Windows(OpWin).InRow-2)*160
		VOff= VOff + (Windows(OpWin).CurCol+Windows(OpWin).InCol-2)*2
	ELSE
		Windows(OpWin).CurCol= Windows(OpWin).CurCol+1
	END IF
LOOP
DEF SEG
IF Ret = -1 THEN
	Windows(OpWin).CurCol=1+(PL/2)
	IF Windows(OpWin).CurRow > Windows(OpWin).Rows-PL THEN
		XST= Windows(OpWin).InCol+(PL/2)-1
		YST= Windows(OpWin).InRow+(PL/2)-1
```

```
                XSB= Windows(OpWin).InCol + Windows(OpWin).Cols-PL-1
                YSB= Windows(OpWin).InRow + Windows(OpWin).Rows-PL-1
                SCROLLUPBOX XST, YST, XSB, YSB, 1, Col
            ELSE
                Windows(OpWin).CurRow= Windows(OpWin).CurRow+1
            END IF
    END IF
    PRINTWINDOW=0
END FUNCTION
```

Nome : **PRNRESET**

È una semplice routine che permette di inviare un segnale hardware di **reset** ad una stampante parallela scelta dall'utente. Il numero di stampante (1 per LPT1, 2 per LPT2 e così via) viene indicato come unico parametro. Sebbene molte stampanti accettino un codice software per ile reset, questa routine permette di effettuare tale operazione anche su stampanti che non lo permettono

```
SUB PRNRESET (PR AS INTEGER)
    Reg.AX=&H100
    Reg.DX=PR-1
    INTERRUPT &H17, Reg, Reg
END SUB
```

Nome : **PRNSTATUS$**

È una funzione alfanumerica che restituisce una stringa il cui contenuto indica lo stato attuale della stampante. La stringa restituita è una tra le seguenti

Off-Line
Spenta
Fine carta
Ok

```
DEFINT A-Z
FUNCTION PRNSTATUS$(PR AS INTEGER)
    Reg.AX=&H200
    Reg.DX=PR-1
    INTERRUPT &H17, Reg, Reg
    ST=Reg.AX / &H100
    SELECT CASE ST
        CASE &H18
            P$="Off-Line"
        CASE &HFFC8
            P$="Spenta"
        CASE &H38
            P$="Fine carta"
        CASE ELSE
            P$="Ok"
    END SELECT
```

```
        PRNSTATUS$=P$+"."
END FUNCTION
```

Nome : **PUTKEY**

È questa una funzione molto particolare che accetta, come parametro, una stringa di un solo carattere e restituisce un valore intero. Essa pone il carattere passato nel buffer della tastiera **come se fosse stato digitato** e ritorna un valore numerico indicante il successo dell'operazione (0), il fatto che il buffer era pieno (1) o che la stringa parametro non era valida (255)

```
DEFINT A-Z
FUNCTION PUTKEY(Ch$)
        IF Ch$<>"" THEN
                Reg.CX=ASC(Ch$)
                Reg.AX=&H500
                INTERRUPT &H16, Reg, Reg
                BUF = Reg.AX AND &HFF
        ELSE
                BUF = &HFF
        END IF
        PUTKEY=BUF
END FUNCTION
```

Nome : **READDIR$**

È la funzione che permette di leggere ed analizzare il contenuto di una directory di un disco. Essa ammette 5 parametri e più precisamente

FILE$ è una stringa che specifica il tipo di file che si vogliono analizzare. Per analizzare l'intera directory corrente del disco di default, basta impostare tale parametro a *.*; se si volesse invece, analizzare tutti i file EXE della directory principale del disco A, sarebbe necessaria la specifica **A:*.EXE**;

ONLYDIRS questo valore intero può assumere valore zero o diverso da zero; quando esso è diverso da zero vengono prese in considerazione soltanto le subdirectory;

NF tale variabile intera, al ritorno dalla funzione, contiene il numero di file trovati su disco.

ND come per la precedente variabile soltanto che questa è usata per contenere il numero delle subdirectory trovate su disco;

FS è una variabile intera lunga il cui contenuto, al rientro della funzione, è eguale al numero totale di bytes occupati da tutti i file trovati.

I dati riguardanti il nome, la grandezza in bytes, la data e l'ora di creazione e gli attributi degli elementi trovati su disco, sono conservati, rispettivamente, nei vettori globali di nome Name$(), Size&(), FDate$(), FTime$() e Typ$().

La seguente porzione di programma mostra come è possibile prelevare tali informazioni dai vettori suddetti e piazzarli in un altro vettore per gestirle con la funzione MCHOICE

```
X$=READDIR$(Spec$, OnlyDirs, NF, ND, TotS&)
IF X$="Ok." THEN
  DIM Elem$(1 TO NF+ND)
  FOR F=1 TO NF+ND
        Elem$(F)=SPACE$(64)
        MID$(Elem$(F), 2)=Names$(F)
        Fsi$= MID$(STR$(Size&(F)), 2)
        MID$(Elem$(F), 17) = SPACE$(10-LEN(Fsi$))+Fsi$
        MID$(Elem$(F), 32) = FDate$(F)
        MID$(Elem$(F), 47) = FTime$(F)
        MID$(Elem$(F), 57) = Typ$(F)
  NEXT F
END IF
Sel=MCHOICE(5, 5, 75, 22, Spec$, Stat$, Col, Elem$(), 64)
```

La funzione READDIR ritorna una stringa di stato che evidenzia il risultato dell'operazione di lettura. Tale stringa può essere una tra le seguenti

File not found.
Illegal path.
Ok.

Esse hanno significato evidente

```
DEFINT A-Z
FUNCTION READDIR$(FILE$, ONLYDIRS, NF, ND, FS AS LONG)
      FSPEC$=FILE$+CHR$(0)
      GETDTA
      Dir.DOSRESERVED=STRING$(21, 0)
      Dir.FILEATT=CHR$(0)
      Dir.FILETIME=STRING$(2, 0)
      Dir.FILEDATE=STRING$(2, 0)
      Dir.FILEIZE=STRING$(4, 0)
      Dir.FILENAME=STRING$(13, 0)
      Dir.FILEDIR=STRING$(85, 0)
      SETDTA VARSEG(Dir), VARPTR(Dir)
      Regx.AX=&H4E00
      Regx.CX=&H2 + &H4 + &H10
      Regx.DX=SADD(FSPEC$)
      Regx.DS=VARSEG(FSPEC$)
      INTERRUPTX &H21, Regx, Regx
      ERC=0
      IF (Regx.FLAGS AND 1) = 1 THEN ERC=Regx.AX
      SELECT CASE ERC
            CASE 2
                  READDIR = "File not found."
                  EXIT FUNCTION
            CASE 3
```

```
                READDIR = “Illegal path.”
                EXIT FUNCTION
            CASE ELSE
                READDIR = “Ok.”
        END SELECT
        NE=0
        NF=0
        ND=0
        FS&=0
        DO WHILE ERC<>18
            CurrAtt$=CONVATTR$(ASC(Dir.FILEATT))
            SWOK=-1
            IF ONLYDIRS<>0 THEN
                IF INSTR(CurrAtt$, “D”) = 0 THEN
                    SWOK=0
                END IF
            END IF
            IF SWOK THEN
                NE=NE+1
                Z=INSTR(Dir.FILENAME, CHR$(0))
                Names$(NE)=LEFT$(Dir.FILENAME, Z-1)+SPACE$(13-Z)
                Typ$(NE)=CONVATTR$(ASC(Dir.FILEATT))
                IF INSTR(Typ$(NE), “D”)>0 THEN
                    ND=ND+1
                    Size&(NE)=0
                ELSE
                    Tmp&=CVL(Dir.FILEIZE)
                    Size&(NE)=Tmp&
                    FS&= FS&+Size&(NE)
                    NF=NF+1
                END IF
                FDate$(NE)=FILEDATE$(Dir.FILEDATE)
                FTime$(NE)=FILETIME$(Dir.FILETIME)
            END IF
            Reg.AX=&H4F00
            INTERRUPT &H21, Reg, Reg
            IF (Reg.FLAGS AND 1) = 1 THEN ERC=Reg.AX
        LOOP
        SETDTA DtaSeg, DtaOff
END FUNCTION
```

Nome : **RESTBOX**

È la routine complementare della funzione SAVEBOX$ ed è usata per visualizzare nella zona video specificata, il contenuto della zona di memoria salvata in precedenza. Per questo motivo essa accetta come parametro, le coordinate video con cui deve operare e la stringa rilasciata dalla SAVEBOX$. La memoria occupata in precedenza dalla SAVEBOX$, viene rilasciata da questa routine

```
SUB RESTBOX(Xt, Yt, Xb, Yb, Handle$)
	IF Handle$="*" THEN
		EXIT SUB
	END IF
	SegArea=VAL(Handle$)
	StOff=INSTR(Handle$, ":")
	OffArea=VAL(MID$(Handle$, StOff+1))
	Largh=Xb-Xt+1
	Lungh=Yb-Yt+1
	VSEG=GETVIDEOSEG
	VOff=(Yt-1)*160+(Xt-1)*2
	IVOFF=VOff
	IAOFF=0
	NRW=Lungh
	DO WHILE NRW>0
		MVDATA SegArea, IAOFF, VSEG, IVOFF, 2*Largh
		IVOFF=IVOFF+160
		IAOFF=IAOFF+2*Largh
		NRW=NRW-1
	LOOP
	Regx.AX=&H4900
	Regx.ES=SegArea
	INTERRUPTX &H21, Regx, Regx
	IF (Regx.FLAGS AND 1)=1 THEN
		EXIT SUB
	END IF
END SUB
```

Nome : **SAVEBOX$**

Questa funzione può essere usata per conservare in memoria una parte della schermata disponibile sul video. Questa possibilità è importante per potere lavorare con le finestre multiple in quanto dà la possibilità di sovrapporre dati sullo schermo senza perdere quelli sottostanti. Tale routine, infatti, viene usata da altre della libreria e più precisamente da quelle che gestiscono le 'windows'.
Essa accetta quattro parametri interi indicanti il numero, rispettivamente, della colonna e della riga dell'angolo in alto e della colonna e della riga dell'angolo in basso della zona video da salvare; viene restituita una stringa contenente alcune informazioni sulla zona di memoria riservata per contenere i dati della zona video e le coordinate della stessa; se un errore si verifica durante l'esecuzione di tale funzione, essa ritorna un solo asterisco

```
FUNCTION SAVEBOX$(Xt, Yt, Xb, Yb)
	Largh=Xb-Xt+1
	Lungh=Yb-Yt+1
	IF Largh<2 OR Lungh<2 THEN
		SAVEBOX$="*"
		EXIT FUNCTION
	END IF
	WSize = CINT(((2*Largh*Lungh) / 16) + .5)
```

```
        Reg.AX=&H4800
        Reg.BX=WSize
        INTERRUPT &H21, Reg, Reg
        IF (Reg.FLAGS AND 1) = 1 THEN
                SAVEBOX$="*"
                EXIT FUNCTION
        END IF
        MemSeg=Reg.AX
        VSEG=GETVIDEOSEG
        VOff= (Yt-1)*160+(Xt-1)*2
        Handle$=STR$(MemSeg)+":0*"
        Handle$=Handle$ + STR$(Xt)+"/"+STR$(Yt)+"/"
        Handle$=Handle$ + STR$(Xb)+"/"+STR$(Yb)
        IVOFF=VOff
        IAOFF=0
        NRW=Lungh
        DO WHILE NRW>0
                MVDATA VSEG, IVOFF, MemSeg, IAOFF, 2*Largh
                IVOFF=IVOFF+160
                IAOFF=IAOFF+2*Largh
                NRW=NRW-1
        LOOP
        SAVEBOX$=Handle$
END FUNCTION
```

Nome : **SCROLLDNBOX**

Insieme alla SCROLLUPBOX, questa routine costituisce uno strumento che è possibile usare per effettuare lo scrolling dei dati di una zona video (modo testo). La routine in questione, scorre **verso il basso** i dati della finestra le cui coordinate (i cui valori vanno tutti decrementati) sono specificate come primi quattro parametri. Il quinto parametro, costituisce il numero di righe che si desidera scorrere e l'ultimo parametro, il codice del colore scelto per le righe nuove che dovranno apparire

```
SUB SCROLLDNBOX(Xt, Yt, Xb, Yb, NL, Col)
        Reg.AX=&H700+NL
        Reg.BX=Col*&H100&
        Reg.CX=Yt*&H100&+Xt
        Reg.DX=Yb*&H100&+Xb
        INTERRUPT &H10, Reg, Reg
END SUB
```

Nome : **SCROLLUPBOX**

Questa routine ha un funzionamento del tutto simile a quello della SCROLLDNBOX ed accetta un numero ed un tipo di parametri del tutto uguale alla precedente. Tuttavia, essa differisce dalla SCROLLDNBOX perché permette di scorrere il testo **verso l'alto**

```
SUB SCROLLUPBOX(Xt, Yt, Xb, Yb, NL, Col)
        Reg.AX=&H600+NL
```

```
	Reg.BX=Col*&H100&
	Reg.CX=Yt*&H100&+Xt
	Reg.DX=Yb*&H100&+Xb
	INTERRUPT &H10, Reg, Reg
END SUB
```

Nome : **SELECTOPZMENU!**

Provvede a gestire uno tra tutti i menu abilitati in precedenza con la sub SETPOPMENU. Tale funzione prevede due parametri, numerici interi, che sono, nell'ordine

- il numero totale dei menu abilitati;
- il numero del menu da gestire

Una volta, correttamente, richiamata, la funzione provvede a fare apparire tutte le opzioni possibili del menu prescelto e rende possibile la scelta di una tra queste con i tasti cursore. con il tasto ESC o Return è possibile scegliere una delle opzioni suddette; la funzione ritorna un valore numerico in singola precisione composto nel seguente modo

- parte intera : numero del menu
- parte decimale : numero dell'opzione

```
FUNCTION SELECTOPZMENU! (MaxMenus, StMenu)
	ActMenu=StMenu
	KCode=-2
	DO WHILE KCode<0
		KCode=GETKEYMENU(ActMenu)
		IF KCode=-2 THEN
			ActMenu= ActMenu+1
			IF ActMenu>MaxMenus THEN
				ActMenu=1
			END IF
		END IF
		IF KCode=-4 THEN
			ActMenu= ActMenu-1
			IF ActMenu=0 THEN
				ActMenu= MaxMenus
			END IF
		END IF
	LOOP
	SELECTOPZMENU! = ActMenu+KCode/10
END FUNCTION
```

Nome : **SETDISK**

Tramite questa routine è possibile settare un nuovo disco corrente per il S.O. È equivalente a fornire, da MS-DOS, l'indicazione del drive sui cui portarsi. L'unico parametro accettato è il nome del drive con cui dovere lavorare in seguito, tramite la corrispondente lettera maiuscola

```
SUB SETDISK (DISK$)
```

```
        Reg.AX=&HE00
        Reg.DX=ASC(UCASE$(DISK$)) - 65
        INTERRUPT &H21, Reg, Reg
END SUB
```

Nome : **SETDTA**

Tale routine, complementare della GETDTA, serve ad impostare, per il S.O., una nuova DTA (Data Transfe Area) e cioè ad impostare un nuovo buffer di memoria per il transito dei dati letti o scritti dai dischi. L'indirizzo dell'area di memoria va indicato con il segmento e l'offset, rispettivamente, primo e secondo parametro della routine stessa

```
SUB SETDTA (DSEG, DOFF)
        Regx.AX=&H1A00
        Regx.DX=DOFF
        Regx.DS=DSEG
        INTERRUPTX &H21, Regx, Regx
END SUB
```

Nome : **SETFILEATTR**

Questa routine è usata per assegnare nuovi attributi ad un determinato file. Il nome del file, eventualmente, completo di drive e path, è fornito come primo parametro mentre, come secondo parametro, viene fornita una stringa contenente, una qualsiasi combinazione delle seguenti lettere maiuscole

R H S A

indicanti i relativi attributi che si vuole abbia il file.

```
SUB SETFILEATTR(FILE$, ATTR$)
        F$=FILE$+CHR$(0)
        Regx.CX=0
        ATTR$=UCASE$(ATTR$)
        IF INSTR(ATTR$, "R")>0 THEN Regx.CX=Regx.CX OR 1
        IF INSTR(ATTR$, "H")>0 THEN Regx.CX=Regx.CX OR 2
        IF INSTR(ATTR$, "S")>0 THEN Regx.CX=Regx.CX OR 4
        IF INSTR(ATTR$, "A")>0 THEN Regx.CX=Regx.CX OR &H20
        Regx.AX=&H4301
        Regx.DX=SADD(F$)
        Regx.DS=VARSEG(F$)
        INTERRUPTX &H21, Regx, Regx
END SUB
```

Nome : **SETMOUSEPOS**

È una delle routines di gestione del mouse e serve a posizionare il cursore dello stesso, in un punto preciso del video. Le coordinate sono fornite, come parametri, alla routine stessa; la prima è la colonna e la seconda la riga. Il cursore viene visualizzato nel punto indicato solo se è stato attivato in precedenza (naturalmente il driver del mouse deve essere stato attivato correttamente)

```
SUB SETMOUSEPOS(POSX, POSY)
	DUM=MOUSE(4, POSX, POSY, 0)
END SUB
```

Nome : **SETPOPMENU**

È una delle routines di gestione dei popup menus; essa viene usata all'inizio per l'inizializzazione degli stessi. In precedenza, devono essere caricati i nomi dei menus e dei sottomenus all'interno dell'array globale di nome **PopMenus$()**; inoltre i codici dei tasti da premere insieme al tasto ALT per richiamare tali menu, devono essere caricati nell'array globale di nome **AltKeysMenu**. La routine accetta come parametro un valore numerico intero rappresentante il numero massimo di menu da attivare. Per tali menu viene usata la prima riga dello schermo in reverse.
In ogni elemento dell'array PopMenus$ deve essere caricato il nome di un menu e dei relativi sottomenus separati da un asterisco. Ad esempio, per il seguente menu

File
Carica
Salva
Esci

va caricata la seguente stringa in un elemento dell'array PopMenu$

File*Carica*Salva*Esci*

```
SUB SETPOPMENU (MaxMenus)
	LOCATE 1, 1
	COLOR 0, 15
	PRINT SPACE$(80);
	LOCATE 1, 2
	FOR Tit=1 TO MaxMenus
		OldX=POS(0)
		TStr$=LEFT$(PopMenus$(Tit), INSTR(PopMenus$(Tit), "*") -1)
		PRINT " "; TStr$; " ";
		LOCATE , OldX+LEN(TStr$)+2
	NEXT Tit
END SUB
```

Nome : **STATUSROW**

È una routine che accetta tre parametri di tipo alfanumerico; queste tre stringhe vengono visualizzate nell'ultima riga dello schermo, in tre aree distinte, la prima di 15, la seconda di 48 e la terza di 15 caratteri. Se una stringa in ingresso è nulla, la relativa area non viene modificata; se è eguale ad uno spazio, l'area viene totalmente cancellata. Le stringhe, qualsiasi sia la loro lunghezza, sono troncate per entrare nelle aree corrispondenti. Questa routine può essere usata, nei programmi applicativi, per visualizzare una riga di stato in cui evidenziare diversi dati

```
SUB STATUSROW(AreaA$, AreaB$, AreaC$)
	COLOR 0, 15
```

```
	LOCATE 25, 1
	SELECT CASE AreaA$
		CASE “”
		CASE “ “
			PRINT SPACE$(15); “3”;
		CASE ELSE
			PRINT LEFT$(AreaA$, 15);
	END SELECT
	LOCATE 25, 17
	SELECT CASE AreaB$
		CASE “”
		CASE “ “
			PRINT SPACE$(48); “3”;
		CASE ELSE
			PRINT LEFT$(AreaB$, 48);
	END SELECT
	LOCATE 25, 66
	SELECT CASE AreaC$
		CASE “”
		CASE “ “
			PRINT SPACE$(15);
		CASE ELSE
			PRINT LEFT$(AreaC$, 15);
	END SELECT
	COLOR 15, 0
END SUB
```

Nome : **TODAY$**

È una semplice, ma utile funzione che non ha alcun parametro e che restituisce una stringa contenente la data del sistema, nel seguente formato

Sss gg Mmm aaaa

in cui Sss è il giorno della settimana, gg il numero del giorno nel mese, Mmm il mese e aaaa l’anno. Ad esempio, per il giorno 3 Novembre 1991, verrebbe restituita dalla funzione la stringa

Dom 3 Nov 1991

```
FUNCTION TODAY$
	TD$=MID$(DATE$,4,2)+”-“+LEFT$(DATE$,2)+”-“+MID$(DATE$,7)
	Gs=DWEEK(TD$)
	TX$=LEFT$(Giorno$(Gs), 3) + STR$(VAL(LEFT$(TD$, 2))) + “ “
	TD$=TX$+LEFT$(Mese$(VAL(MID$(TD$,4,2))), 3)+STR$(VAL(MID$(TD$, 7)))
	TODAY$=TD$
END FUNCTION
```

7.2.3 Uso della libreria BPLUS : programma di esempio

Come programma di esempio che usi la libreria BPLUS, si è pensato di realizzarne uno in grado di sostituire MS-DOS avvalendosi di menu, come del resto fa la Shell di MS-DOS 4.0 o di MS-DOS 5.0. Naturalmente, per motivi di spazio e di tempo, tale programma (chiamato Quick DOS, **QDOS** in breve) non sarà completo di tutto quello che è possibile e necessario fare con una Shell di MS-DOS, ma rappresenta comunque un buon punto di partenza per chi vorrà completare l'opera.
Tramite tale programma, che sfrutta finestre e popup menu, è possibile effettuare le seguenti operazioni

- Cambio del disco di default
- Formattazione di un disco
- Copia di uno o più file
- Spostamento di uno o più file
- Cancellazione di uno o più file
- Cambio del nome di uno o più file
- Ricerca di dati all'interno di file
- Visualizzazione del contenuto di file
- Comparazione di due file
- Modifica attributi di uno o più file
- Creazione di una subdirectory
- Cambio della subdirectory di default
- Rimozione di una subdirectory
- Modifica della data di sistema
- Modifica dell'ora di sistema
- Visualizzazione della memoria disponibile
- Visualizzazione delle caratteristiche dei dischi
- Shell verso MS-DOS
- Informazioni sul programma

È evidente che sarebbe utile potere anche copiare dei dischi, avere a disposizione un editor per modificare i file e così via. Non esiste limite alle funzioni che si potrebbero aggiungere a QDOS per migliorarlo, ma questa operazione si riserva al lettore.

```
‘
‘ (Main Module QDOS (c) versione 2.0.)
‘ (Antonio Giuliana, Monreale 1991)
‘ (È necessaria la libreria BPLUS)
‘
DEFINT A-Z
DECLARE SUB About()
DECLARE SUB ChangeDir(SelFile$)
DECLARE SUB ChangeDisk()
DECLARE SUB ChgAttr(Select$)
DECLARE SUB ClAnalyze(Hdl)
DECLARE SUB Compare(Select$)
```

```
DECLARE SUB Copy(Select$, IsMove)
DECLARE SUB Delete(Select$)
DECLARE SUB DFormat()
DECLARE SUB DiskList()
DECLARE SUB DOS()
DECLARE SUB MakeDir()
DECLARE SUB MemList()
DECLARE SUB ModDate()
DECLARE SUB ModTime()
DECLARE FUNCTION OpAnalyze()
DECLARE SUB RemoveDir(SelFile$)
DECLARE SUB Rename(Select$)
DECLARE SUB Search(Select$)
DECLARE SUB VisDir(Spec$, Select$, OnlyDirs)
DECLARE SUB VisFile(Select$)
' $INCLUDE: 'BPLUS.INC'
' $INCLUDE: 'TYPES.INC'
' $INCLUDE: 'COMMON.INC'
' $INCLUDE: 'DOS.INC'
' $INCLUDE: 'POPMENU.INC'
' $INCLUDE: 'WIN.INC'
' $INCLUDE: 'DIR.INC'
' $INCLUDE: 'DATETIME.INC'
ON ERROR GOTO ErTrap
DIM SHARED CDisk$
DIM SHARED CPath$
OldX=POS(0)
OldY=CSRLIN
LOCATE 1, 1
Menus=7
RESTORE PopMenu
FOR MM=1 TO Menus
        READ PopMenus$(MM)
        READ AltKeysMenus(MM)
NEXT MM
W1=OPENWINDOW(1, 1, 80, 25, 58, 2)
SETPOPMENU Menus
STATUSROW " ", " ", " "
About
LOCATE 1, 69
COLOR 0, 6
PRINT " QDOS 2.0. "
DO
   CDisk$=CURRDISK$
   CPath$=CURRPATH$(CDisk$)
   CPath$=LEFT$(CPath$, LEN(CPath$)-1)
   BFree&=DISKFREE&(CDisk$)
   IF BFree&<100000 THEN
```

```
    BFree$=STR$(BFree&)
ELSE
    BFree$=STR$(INT(BFree&/1024&)) + " K"
END IF
Fie1$=BFree$+ " Free"+SPACE$(10)
STATUSROW Fie1$, CPath$+SPACE$(46), TODAY$+SPACE$(5)
AKey$=""
DO WHILE AKey$=""
    AKey$=INKEY$
    COLOR 0, 15
    LOCATE 1, 60
    PRINT TIME$;
LOOP
IF LEN(AKey$)=2 THEN
    KCode=ASC(MID$(AKey$, 2))
    SelMenu=0
    FOR MM=1 TO Menus
        IF AltKeysMenu(MM)=KCode THEN
            SelMenu=MM
            EXIT FOR
        END IF
    NEXT MM
    IF SelMenu>0 THEN
        SelOpzMenu! = SELECTOPZMENU!(Menus, SelMenu)
    ELSE
        SelOpzMenu! = 0
    END IF
    IF SelOpzMenu! <> INT(SelOpzMenu!) THEN
        SubMenu = (SelOpzMenu! - INT(SelOpzMenu!))*10
        SELECT CASE INT(SelOpzMenu!)
            CASE 1
                SELECT CASE SubMenu
                    CASE 1
                        ChangeDisk
                    CASE 2
                        DFormat
                    CASE ELSE
                END SELECT
            CASE 2
                IF RIGHT$(CPath$ ,1)="\" THEN
                    Spec$=CPath$+"*.*"
                ELSE
                    Spec$=CPath$+"\*.*"
                END IF
                VisDir Spec$, SelFile$, 0
                SELECT CASE SubMenu
                    CASE 1
                        Copy SelFile$, 0
```

```
        CASE 2
            Copy SelFile$, -1
        CASE 3
            Delete SelFile$
        CASE 4
            Rename SelFile$
        CASE 5
            Search SelFile$
        CASE 6
            VisFile SelFile$
        CASE 7
            Compare SelFile$
        CASE 8
            ChgAttr SelFile$
        CASE ELSE
    END SELECT
CASE 3
    IF SubMenu>1 THEN
        IF RIGHT$(CPath$, 1)="\" THEN
            Spec$=CPath$+"*.*"
        ELSE
            Spec$=CPath$+"\*.*"
        END IF
        VisDir Spec$, SelFile$, 1
    ENDIF
    SELECT CASE SubMenu
        CASE 1
            MakeDir
        CASE 2
            ChangeDir SelFile$
        CASE 3
            RemoveDir SelFile$
        CASE ELSE
    END SELECT
CASE 4
    SELECT CASE SubMenu
        CASE 1
            ModDate
        CASE 2
            ModTime
        CASE ELSE
    END SELECT
CASE 5
    SELECT CASE SubMenu
        CASE 1
            MemList
        CASE 2
            DiskList
```

```
                CASE ELSE
            END SELECT
        CASE 6
            SELECT CASE SubMenu
                CASE 1
                    DOS
                CASE 2
                    LOCATE OldY, OldX
                    EXIT DO
                CASE ELSE
            END SELECT
        CASE 7
            About
        CASE ELSE
      END SELECT
    END IF
  END IF
LOOP
ST=CLEARWINDOW(W1, 58)
ST=CLOSEWINDOW(W1)
END
PopMenu:
DATA Disco*Cambia*Formatta*, 32
DATA File*Copia*Muove*Cancella*Rinomina*Ricerca*Visualizza*Compara*Attributi*, 33
DATA Dir*Crea*Cambia*Rimuove*, 23
DATA Modifica*Data*Orario*, 50
DATA Risorse*Memoria*Dischi*, 19
DATA Quit*DOS*Fine*, 16
DATA Info*Su Quick DOS …*, 49
ErTrap:
  IF ERL=111 OR ERL=222 THEN
    BEEP
    RESUME NEXT
  END IF
  ON ERROR GOTO 0
REM $DYNAMIC
SUB About
  T1$="Quick Basic BPLUS Library"
  T2$=" - Antonio Giuliana, Monreale 1991 – "
  W2 = OPENWINDOW(20, 8, 60, 18, 66, 1)
  ST=CENTREWIN(W2, "Quick DOS", 79, 0, 5)
  ST=CENTREWIN(W2, "Vers. 2.0. - Antonio Giuliana", 71, 0, 7)
  ST=CENTREWIN(W2, "Monreale 1991", 71, 0, 8)
  A$=INPUT$(1)
  ST=CLOSEWINDOW(W2)
END SUB
REM $STATIC
SUB ChangeDir(SelFile$)
```

```
    El=VAL(MID$(SelFile$, 2))
    NDir$=Names$(El)
    IF El>0 THEN
        SHELL "CD " + NDir$ + " >NUL"
    END IF
END SUB
SUB ChangeDisk
    WCH=OPENWINDOW(10, 10, 40, 14, 66, 2)
    ST=CURSORWINDOW(WCH, 3, 3)
    ST=PRINTWINDOW(WCH, "Nuovo disco selezionato :", 79, 0)
    Exc=CINPUT(38, 12, 1, CDisk$, Drv$, 79, 0, 250)
    Drv$=UCASE$(Drv$)
    IF Exc<>0 AND LEN(Drv$)>0 THEN
        IF Drv$ >= "A" AND Drv$ <= "Z" THEN
            SETDISK (Drv$)
        ELSE
            BEEP
        END IF
    END IF
    ST=CLOSEWINDOW(WCH)
END SUB
SUB ChgAttr(Select$)
    Col=42
    IF LEN(Select$)>0 THEN
        Chg=OPENWINDOW(5, 5, 75, 18, Col, 2)
        ST=CENTREWIN(Chg, "Change Attributes …", Col, 0, 3)
        ST=CENTREWIN(Chg, STRING$(21, CHR$(196)), Col, 0, 4)
        APos=INSTR(Select$, "*")
        DO WHILE APos>0
            El=VAL(MID$(Select$, APos+1))
            NoOk=0
            NoOk=NoOk+INSTR(Typ$(El), "V")
            Ftc$=RTRIM$(Names$(El))
            APos=INSTR(APos+1, Select$, "*")
            IF NoOk =0 THEN
            Bs$=""
            IF RIGHT$(CPath$, 1)<>"\" THEN
            Bs$="\"
            END IF
            ST=CENTREWIN(Chg, "Changing …", Col, 0, 6)
            ST=CENTREWIN(Chg, SPACE$(40), Col, 0, 7)
            ST=CENTREWIN(Chg, CPath$+Bs$+Ftc$, Col, 0, 7)
            ST=CENTREWIN(Chg, SPACE$(40), Col, 0, 9)
            ST=CENTREWIN(Chg, "Actual Attributes : "+Typ$(El), Col, 0, 9)
            ST=CENTREWIN(Chg, "New Attributes", Col, 0, 11)
            DUM=CINPUT(38, 16, 4, "", NAtt$, Col, 0, 250)
            SETFILEATTR Ftc$, NAtt$
            END IF
```

```
        LOOP
        ST=CLOSEWINDOW(Chg)
    END IF
END SUB
SUB ClAnalyze(Hdl)
    ST=CLOSEWINDOW(Hdl)
END SUB
SUB Compare(Select$)
    IF LEN(Select$)>0 THEN
        DIM Ftc$(1 TO 2)
        Fc=1
        APos=INSTR(Select$, "*")
        DO WHILE APos>0
            El=VAL(MID$(Select$, APos+1))
            NoOk=0
            NoOk=NoOk+INSTR(Typ$(El), "D")
            NoOk=NoOk+INSTR(Typ$(El), "V")
            Ftc$=RTRIM$(Names$(El))
            APos=INSTR(APos+1, Select$, "*")
            IF NoOk=0 THEN
                Ftc$(Fc)=Ftc$
                IF Fc=2 THEN
                    EXIT DO
                END IF
                Fc=Fc+1
            END IF
        LOOP
        IF Fc=2 THEN
            Col=42
            Cpr=OPENWINDOW(5, 5, 75, ,16, Col, 2)
            ST=CENTREWIN(Cpr, "Compare file …", Col, 0, 3)
            ST=CENTREWIN(Cpr, STRING$(17, CHR$(196)), Col, 0, 4)
            Fil1=FREEFILE
            OPEN "B", Fil1, Ftc$(1)
            Fil2=FREEFILE
            OPEN "B", Fil2, Ftc$(2)
            Lf1&=LOF(Fil1)
            Lf2&=LOF(Fil2)
            ST=CENTREWIN(Cpr, "Comparing file", Col, 0, 6)
            ST=CENTREWIN(Cpr, Ftc$(1) + " to " + Ftc$(2), Col, 0, 7)
            Buf1$=SPACE$(512)
            Buf2$=SPACE$(512)
            FIdx&=1
            IF Lf1$=Lf2$ THEN
                DO
                    SEEK #Fil1, FIdx&
                    SEEK #Fil2, FIdx&
                    GET #Fil1, , Buf1$
```

```
                    GET #Fil2, , Buf2$
                    ST=CENTREWIN(Cpr, “( Offset $”+HEX$(FIdx&)+” )”, Col, 0, 9)
                    IF Buf1$<>Buf2$ THEN
                        BEEP
                        ST=CENTREWIN(Cpr, “Esistono delle differenze tra i file.”, Col, 0, 9)
                        EXIT DO
                    END IF
                    IF FIdx&>Lf1& THEN
                        BEEP
                        ST=CENTREWIN(Cpr, “I file sono uguali.”, Col, 0, 9)
                        EXIT DO
                    END IF
                    FIdx& = FIdx&+512
                LOOP
            ELSE
                BEEP
                ST=CENTREWIN(Cpr, “I file hanno dimensioni diverse.”, Col, 0, 9)
            END IF
            Buf1$=””
            Buf2$=””
            CLOSE
            A$=INPUT$(1)
            ST=CLOSEWINDOW(Cpr)
        END IF
    END IF
END SUB
SUB Copy(Select$, IsMove)
    Col=42
    IF LEN(Select$)>0 THEN
        Cpy=OPENWINDOW(5, 5, 75, 16, Col, 2)
        Op$=”Copy”
        IF IsMove THEN
            Op$=”Move”
        END IF
        ST=CENTREWIN(Cpy, Op$ + “ file to Path …”, Col, 0, 3)
        ST=CENTREWIN(Cpy, STRING$(22, CHR$(196)), Col, 0, 4)
        Exc=CINPUT(8, 11, 64, CURRDISK$ + “\”, TPath$, Col, 0, 250)
        IF Exc<>>0 THEN
            TPath$=UCASE$(TPath$)
            LOCATE 11, 8
            PRINT TPath$
            APos=INSTR(Select$, “*”)
            DO WHILE APos>0
                El=VAL(MID$(Select$, APos+1))
                NoOk=0
                NoOk=NoOk+INSTR(Typ$(El), “D”)
                NoOk=NoOk+INSTR(Typ$(El), “V”)
                Ftc$=RTRIM$(Names$(El))
```

```
                APos=INSTR(APos+1, Select$, "*")
                IF NoOk = 0 THEN
                    Op$="Copying"
                    IF IsMove THEN
                        Op$="Moving"
                    END IF
                    Dest$=TPath$
                    IF RIGHT$(Dest$, 1) = "\" THEN
                        Dest$=Dest$+Ftc$
                    END IF
                    ST=CENTREWIN(Cpy, Op$ + " file", Col, 0, 9)
                    ST=CENTREWIN(Cpy, Ftc$ + " to " + Dest$, Col, 0, 10)
                    SHELL "Copy "+Ftc$ + " " + Dest$ + " >NUL"
                    IF IsMove THEN
                        SHELL "Del "+Ftc$ + " >NUL"
                    END IF
                    ST=CENTREWIN(Cpy, SPACE$(40), Col, 0, 9)
                    ST=CENTREWIN(Cpy, SPACE$(40), Col, 0, 10)
                END IF
            LOOP
        END IF
        ST=CLOSEWINDOW(Cpy)
    END IF
END SUB
SUB Delete(Select$)
    Col=42
    IF LEN(Select$)>0 THEN
        Del=OPENWINDOW(5, 5, 75, 16, Col, 2)
        ST=CENTREWIN(Del, "Delete file …", Col, 0, 3)
        ST=CENTREWIN(Del, STRING$(16, CHR$(196)), Col, 0, 4)
        APos=INSTR(Select$, "*")
        DO WHILE APos>0
            El=VAL(MID$(Select$, APos+1))
            NoOk=0
            NoOk=NoOk+INSTR(Typ$(El), "D")
            NoOk=NoOk+INSTR(Typ$(El), "V")
            Ftc$=RTRIM$(Names$(El))
            APos=INSTR(APos+1, Select$, "*")
            IF NoOk = 0 THEN
                Bs$=""
                IF RIGHT$(CPath$, 1) <> "\" THEN
                    Bs$= "\"
                END IF
                ST=CENTREWIN(Del, "Deleting file", Col, 0, 9)
                ST=CENTREWIN(Del, CPath$ + Bs$ + Ftc$, Col, 0, 10)
                SHELL "Del "+Ftc$ + " >NUL"
                ST=CENTREWIN(Del, SPACE$(40), Col, 0, 9)
                ST=CENTREWIN(Del, SPACE$(40), Col, 0, 10)
```

```
        END IF
      LOOP
      ST=CLOSEWINDOW(Del)
    END IF
END SUB
SUB DFormat
    Col=47
    WFm=OPENWINDOW(10, 6, 70, 20, Col, 2)
    ST=CENTREWIN(WFm, “Formattazione Dischetti”, Col, 0, 3)
    ST=CENTREWIN(WFm, STRING$(13, CHR$(196)), Col, 0, 4)
    ST=CURSORWINDOW(WFm, 8, 10)
    ST=PRINTWINDOW(WFm, “Dischetto da formattare :”, Col, 0)
    ST=CURSORWINDOW(WFm, 10, 10)
    ST=PRINTWINDOW(WFm, “Parametri da passare    :”, Col, 0)
    Exc=CINPUT(47, 13, 1, “A”, Df$, Col, 0, 250)
    Df$=UCASE$(Df$)
    IF Exc <> 0 AND LEN(Df$)>0 THEN
      IF Df$=”A” OR Df$=”B” THEN
        Exc=CINPUT(47, 15, 20, “”, Par$, Col, 0, 250)
        IF Exc<>0 THEN
          SHELL “Format “ + Df$ + “: “ + Par$ + “<QDOS.KB1 >NUL”
          ST=CENTREWIN(WFm, “Formattazione Completata.”, Col, 0, 12)
          A$=INPUT$(1)
        END IF
      ELSE
        BEEP
      END IF
    END IF
    ST=CLOSEWINDOW(WFm)
END SUB
SUB DiskList
    Col=47
    WX=OPENWINDOW(10, 6, 70, 20, Col, 2)
    ST=CENTREWIN(WX, “Elenco Dischi”, Col, 0, 3)
    ST=CENTREWIN(WX, STRING$(13, CHR$(196)), Col, 0, 4)
    Row=6
    EndDisk=0
    HDisk=&H7F
    FDisk=-1
    FOR Drive=1 TO 26
      Disk$=” “+CHR$(Drive+64)+”: “
      DTy$=” “
      Reg.AX=&H4408
      Reg.BX=Drive
      INTERRUPT &H21, Reg, Reg
      IF (Reg.FLAGS AND 1)=1 THEN
        EXIT FOR
      END IF
```

```
SELECT CASE Reg.AX
    CASE 0
        DTyp$=DTyp$+ "Floppy Disk Drive "
        FDisk=FDisk+1
        CDisk=FDisk
    CASE 1
        DTyp$=DTyp$+ "Hard Disk Drive "
        HDisk=HDisk+1
        CDisk=HDisk
    CASE ELSE
END SELECT
IF EndDisk=0 THEN
    Reg.AX=&H800
    Reg.DX=CDisk
    INTERRUPT &H13, Reg, Reg
    IF (Reg.FLAGS AND 1)=1 THEN
        EndDisk=-1
    END IF
    IF EndDisk=0 THEN
        IF CDisk<&H80 THEN
            SELECT CASE Reg.BX
                CASE 1
                    DTy$=DTy$ + "5" + CHR$(34) + " , DD"
                CASE 2
                    DTy$=DTy$ + "5" + CHR$(34) + " , HD"
                CASE 3
                    DTy$=DTy$ + "3" + CHR$(34) + " + DD"
                CASE 4
                    DTy$=DTy$ + "3" + CHR$(34) + " + HD"
                CASE ELSE
                    DTy$=DTy$ + "Sconosciuto"
            END SELECT
        END IF
    END IF
END IF
IF Row>13 THEN
    ABEEP
    Row=6
    COLOR 15, 2
    LOCATE 20, 45
    PRINT "Ancora … ";
    A$=INPUT$(1)
    LOCATE 20, 45
    PRINT STRING$(12, CHR$(205));
    CLEARBOX 23, 9, 55, 18, Col
END IF
IF Drive>2 THEN
    BFree&=DISKFREE&(MID$(Disk$, 2))
```

```
            IF BFree&<100000 THEN
                BFree$=”(“ + STR$(BFree&) + “ Free )”
            ELSE
                BFree$=”(“ + STR$(INT(BFree& / 1024&)) + “ K Free )”
            END IF
        ELSE
            BFree$=””
        END IF
        ST=CURSORWINDOW(WX, Row, 5)
        ST=PRINTWINDOW(WX, Disk$, 63, 0)
        ST=CURSORWINDOW(WX, Row, 10)
        ST=PRINTWINDOW(WX, DTy$ + “ “ + BFree$, Col, 0)
        Row=Row+1
    NEXT Drive
    A$=INPUT$(1)
    ST=CLOSEWINDOW(WX)
END SUB
SUB DOS
    Sh$=SAVEBOX$(1, 1, 80, 25)
    CLS
    PRINT “Type EXIT FOR QDOS …”
    PRINT
    SHELL
    RESTBOX 1, 1, 80, 25, Sh$
END SUB
SUB MakeDir
    WMD=OPENWINDOW(10, 10, 70, 14, 66, 2)
    ST=CURSORWINDOW(WMD, 3, 3)
    ST=PRINTWINDOW(WMD, “Nuova directory : ”, 79, 0)
    Exc=CINPUT(30, 12, 12, “”, NDir$, 79, 0, 250)
    IF Exc<>0 AND LEN(NDir$)>0 THEN
        SHELL “MD “+NDir$+” >NUL”
    ELSE
        BEEP
    END IF
    ST=CLOSEWINDOW(WMD)
END SUB
SUB MemList
    Col=47
    WX=OPENWINDOW(10, 6, 70, 20, Col, 2)
    ST=CENTREWIN(WX, “Memoria di sistema”, Col, 0, 3)
    ST=CENTREWIN(WX, STRING$(18, CHR$(196)), Col, 0, 4)
    EXPEXT TEXPMEM, FEXPMEM, FEXTMEM
    CVMEM&=FRE(-1)
    ST=CURSORWINDOW(WX, 7, 8)
    ST=PRINTWINDOW(WX, “Memoria convenzionale libera =” + STR$(CVMEM&), Col, 0)
    ST=CURSORWINDOW(WX, 9, 8)
    ST=PRINTWINDOW(WX, “Memoria espansa totale =” + STR$(TEXPMEM*1024&), Col, 0)
```

```
    ST=CURSORWINDOW(WX, 10, 8)
    ST=PRINTWINDOW(WX, "Memoria espansa libera =" + STR$(FEXPMEM*1024&), Col, 0)
    ST=CURSORWINDOW(WX, 11, 8)
    ST=PRINTWINDOW(WX, "Memoria estesa libera =" + STR$(FEXTMEM*1024&), Col, 0)
    TMem&=(FEXTMEM+FEXPMEM)*1024&+CVMEM&
    ST=CURSORWINDOW(WX, 13, 8)
    ST=PRINTWINDOW(WX, "Totale Memoria disponibile =" + STR$(TMem&)+SPACE$(5), 127, 0)
    FOR Rw=7 TO 13
        IF Rw<>8 AND Rw<>12 THEN
            ST=CURSORWINDOW(WX, Rw, 48)
            IF Rw=13 THEN
                Col=127
            END IF
            ST=PRINTWINDOW(WX, " bytes", Col, 0)
        END IF
    NEXT Rw
    AKey$=INPUT$(1)
    ST=CLOSEWINDOW(WX)
END SUB
SUB ModDate
    Col=47
    WMD=OPENWINDOW(10, 6, 70, 20, Col, 2)
    ST=CENTREWIN(WMD, "Modifica Data", Col, 0, 3)
    ST=CENTREWIN(WMD, STRING$(13, CHR$(196)), Col, 0, 4)
    ST=CURSORWINDOW(WMD, 8, 10)
    ST=PRINTWINDOW(WMD, "Nuova data (mm/gg/aaaa) :", Col, 0)
    Exc=CINPUT(47, 13, 10, "", NDate$, Col, 0, 250)
    IF Exc<>0 THEN
111  DATE$=MID$(NDate$, 4, 3)+LEFT$(NDate$, 3)+RIGHT$(NDate$, 4)
    END IF
    ST=CLOSEWINDOW(WMD)
END SUB
SUB ModTime
    Col=47
    WMD=OPENWINDOW(10, 6, 70, 20, Col, 2)
    ST=CENTREWIN(WMD, "Modifica Orario", Col, 0, 3)
    ST=CENTREWIN(WMD, STRING$(13, CHR$(196)), Col, 0, 4)
    ST=CURSORWINDOW(WMD, 8, 10)
    ST=PRINTWINDOW(WMD, "Nuovo orario (hh:mm:ss) :", Col, 0)
    Exc=CINPUT(47, 13, 8, "", NTime$, Col, 0, 250)
    IF Exc<>0 THEN
        IF LEN(NTime$)=2 THEN
            NTime$=NTime$+":00:00"
        ELSE
            NTime$=NTime$+":00"
        END IF
222   TIME$=NTime$
    END IF
```

```
    ST=CLOSEWINDOW(WMD)
END SUB
FUNCTION OpAnalyze
    Col=42
    AN=OPENWINDOW(25, 10, 55, 14, Col, 2)
    ST=CENTREWIN(AN, " Analyze …", 175, 0, 3)
    OpAnalyze=AN
END FUNCTION
SUB RemoveDir(SelFile$)
    LOCATE 5, 5
    Col=42
    IF LEN(SelFile$)>0 THEN
        Rmd=OPENWINDOW(5, 5, 75, 16, Col, 2)
        ST=CENTREWIN(Rmd, "Remove directory …", Col, 0, 3)
        ST=CENTREWIN(Rmd, STRING$(20, CHR$(196)), Col, 0, 4)
        APos=INSTR(SelFile$, "*")
        DO WHILE APos>0
            El=VAL(MID$(SelFile$, APos+1))
            Dtr$=RTRIM$(Names$(El))
            APos=INSTR(APos+1, SelFile$, "*")
            ST=CENTREWIN(Rmd, "Removing directory", Col, 0, 9)
            ST=CENTREWIN(Rmd, Dtr$, Col, 0, 10)
            IF Dtr$<>"." AND Dtr$<>".." THEN
                SHELL "RD "+Dtr$+" >NUL"
            END IF
            ST=CENTREWIN(Rmd, SPACE$(40), Col, 0, 9)
            ST=CENTREWIN(Rmd, SPACE$(40), Col, 0, 10)
        LOOP
        ST=CLOSEWINDOW(Rmd)
    END IF
END SUB
SUB Rename(Select$)
    Col=42
    IF LEN(Select$)>0 THEN
        Ren=OPENWINDOW(5, 5, 75, 16, Col, 2)
        ST=CENTREWIN(Ren, "Rename file …", Col, 0, 3)
        ST=CENTREWIN(Ren, STRING$(16, CHR$(196)), Col, 0, 4)
        APos=INSTR(Select$, "*")
        DO WHILE APos>0
            El=VAL(MID$( Select$, APos+1))
            NoOk=0
            NoOk=NoOk+INSTR(Typ$(El), "D")
            NoOk=NoOk+INSTR(Typ$(El), "V")
            Ftc$=RTRIM$(Names$(El))
            APos=INSTR(APos+1, Select$, "*")
            IF NoOk=0 THEN
                ST=CENTREWIN(Ren, "Old File : "+Ftc$+" to …", Col, 0, 6)
                DUM=CINPUT(35, 11, 12, Ftc$, NName$, Col, 0, 250)
```

```
            NName$=UCASE$(NName$)
            LOCATE 11, 35
            PRINT NName$;
            ST=CENTREWIN(Ren, "Renaming file from", Col, 0, 9)
            ST=CENTREWIN(Ren, Ftc$ + " to " + NName$, Col, 0, 10)
            SHELL "Ren "+Ftc$+" "+NName$+" >NUL"
            ST=CENTREWIN(Ren, SPACE$(40), Col, 0, 9)
            ST=CENTREWIN(Ren, SPACE$(40), Col, 0, 10)
        END IF
    LOOP
    ST=CLOSEWINDOW(Ren)
  END IF
END SUB
SUB Search(Select$)
  Col=42
  IF LEN(Select$)>0 THEN
    Src=OPENWINDOW(5, 5, 75, 20, Col, 2)
    ST=CENTREWIN(Src, "Search data into file …", Col, 0, 3)
    ST=CENTREWIN(Src, STRING$(26, CHR$(196)), Col, 0, 4)
    DUM=CINPUT(30, 11, 20, "", SData$, Col, 0, 250)
    Ex=0
    APos=INSTR(Select$, "*")
    DO WHILE APos>0
        El=VAL(MID$(Select$, APos+1))
        NoOk=0
        NoOk=NoOk+INSTR(Typ$(El), "D")
        NoOk=NoOk+INSTR(Typ$(El), "V")
        Ftc$=RTRIM$(Names$(El))
        APos=INSTR(APos+1, Select$, "*")
        IF NoOk=0 THEN
            ST=CENTREWIN(Src, "Searching in file", Col, 0, 9)
            ST=CENTREWIN(Src, Ftc$, Col, 0, 10)
            FS=FREEFILE
            OPEN "B", FS, Ftc$
            Lf& = LOF(FS)
            Nxt&=1
            Buf$=SPACE$(LEN(SData$))
            DO WHILE Lf&>0
                SEEK #FS, Nxt&
                GET #FS, , Buf$
                IF SData$=Buf$ THEN
                    ST=CENTREWIN(Src, "Found at offset "+ HEX$(Nxt&), Col, 0, 12)
                    BEEP
                    VBuf$=SPACE$(60)
                    Pv&=Nxt&-30
                    IF Pv&<1 THEN
                        Pv&=1
                    END IF
```

```
                SEEK #FS, Pv&
                GET #FS, , VBuf$
                ST=CENTREWIN(Src, VBuf$, 47, 0, 14)
                VBuf$=""
                A$=INPUT$(1)
                IF A$=CHR$(27) THEN
                    Ex=-1
                    EXIT DO
                END IF
                ST=CENTREWINDOW(Src, SPACE$(40), Col, 0, 12)
            END IF
            Nxt&=Nxt&+1
            Lf&=Lf&-1
        LOOP
        CLOSE FS
        IF Ex THEN
            EXIT DO
        END IF
        ST=CENTREWIN(Src, SPACE$(40), Col, 0, 9)
        ST=CENTREWIN(Src, SPACE$(40), Col, 0, 10)
      END IF
    LOOP
    ST=CLOSEWINDOW(Src)
  END IF
END SUB
SUB VisDir(Spec$, Select$, OnlyDirs)
  Lyz=OpAnalyze
  Col=42
  X$=READDIR$(Spec$, OnlyDirs, NF, ND, TotS&)
  IF X$="Ok." THEN
    OkFile=-1
    Stat$="Disk : " + DISKTYPE$(Spec$)
    Stat$=Stat$+ " z" + STR$(DISKFREE&(Spec$))+ " Free"
    Stat$=Stat$+ " z" + STR$(TotS&)+ " occ. da" + STR$(NF+ND)+ " File"
    IF NF+ND>0 THEN
      DIM Elem$(1 TO NF+ND)
      FOR F=1 TO NF+ND
        Elem$(F)=SPACE$(64)
        MID$(Elem(F), 2)=Name$(F)
        Fsi$=MID$(STR$(Size&(F)), 2)
        MID$(Elem$(F), 17)=SPACE$(10-LEN(Fsi$))+Fsi$
        MID$(Elem$(F), 32)=FDate$(F)
        MID$(Elem$(F), 47)=FTime$(F)
        MID$(Elem$(F), 57)=Typ$(F)
      NEXT F
    ELSE
      OkFile=0
      DIM Elem$(1 TO 1)
```

```
            Elem$(1)=SPACE$(20)+"No File or Directory found."+SPACE$(14)
        END IF
        Sel=MCHOICE(5, 5, 75, 22, Spec$, Stat$, Col, Elem$(), 64)
    END IF
    ClAnalyze(Lyz)
    Select$=""
    IF Sel<>0 AND OkFile THEN
        FOR F=1 TO NF+ND
            IF MID$(Elem$(F), 64, 1)=CHR$(251) THEN
                Select$=Select$ + "*" + STR$(F)
            END IF
        NEXT F
    END IF
END SUB
SUB VisFile(Select$)
    IF LEN(Select$)>0 THEN
        DIM Righe(1 TO 19) AS STRING * 70
        Col=42
        Vis=OPENWINDOW(2, 3, 79, 23, Col, 2)
        Ex=0
        APos=INSTR(Select$, "*")
        DO WHILE APos>0
            El=VAL(MID$(Select$, APos+1))
            NoOk=0
            NoOk=NoOk+INSTR(Typ$(El), "S")
            NoOk=NoOk+INSTR(Typ$(El), "H")
            NoOk=NoOk+INSTR(Typ$(El), "D")
            NoOk=NoOk+INSTR(Typ$(El), "V")
            Ftc$=RTRIM$(Names$(El))
            APos=INSTR(APos+1, Select$, "*")
            IF NoOk=0 THEN
                COLOR 15, Col \ 16
                LOCATE 23, 10
                PRINT " PgUp PgDn "
                LOCATE 23, 50
                PRINT " Ret Esc ";
                FS=FREEFILE
                OPEN "B", FS, Ftc$
                Lf&=LOF(FS)
                FIdx&=1
                DO
                    COLOR 15, Col \ 16
                    LOCATE 3, 5
                    PRINT " File : "; Ftc$; "  at offset $"; HEX$(FIdx&);
                    COLOR Col MOD 16, Col \ 16
                    PRINT STRING$(10, 205);
                    SEEK #FS, FIdx&
                    FOR Rw=1 TO 19
```

```
                Righe(Rw)=SPACE$(70)
                GET #FS, , Righe(Rw)
                ST=CURSORWINDOW(Vis, Rw+1, 3)
                ST=PRINTWINDOW(Vis, Righe(Rw), Col, 0)
            NEXT Rw
            Ak$=""
            DO WHILE Ak$=""
                Ak$=INKEY$
            LOOP
            SELECT CASE LEN(Ak$)
                CASE 1
                    Ak=ASC(Ak$)
                    SELECT CASE Ak
                        CASE 13
                            BEEP
                            EXIT DO
                        CASE 27
                            Ex=-1
                            EXIT DO
                        CASE ELSE
                    END SELECT
                CASE 2
                    Ak=ASC(MID$(Ak$, 2))
                    SELECT CASE Ak
                        CASE 73
                            FIdx&= FIdx&-1330&
                            IF FIdx&<1 THEN
                                FIdx&=1
                            END IF
                        CASE 81
                            FIdx&= FIdx&+1330&
                            IF FIdx&>Lf& THEN
                                FIdx&=Lf&-1330&
                            END IF
                            IF FIdx&<1 THEN
                                FIdx&=1
                            END IF
                        CASE ELSE
                    END SELECT
                CASE ELSE
            END SELECT
        LOOP
        CLOSE FS
        IF Ex THEN
            EXIT DO
        END IF
    END IF
LOOP
```

```
        ST=CLOSEWINDOW(Vis)
    END IF
END SUB
```

La compilazione di QDOS può essere fatta con il comando

BC QDOS;

mentre, per il link, si darà il seguente comando

LINK /NOE QDOS,,NUL,BPLUS MLIBCE

tenendo presente che le due librerie, BPLUS.LIB e MLIBCE.LIB (del compilatore C), devono essere poste nella directory di lavoro oppure, nella directory indicata nella variabile d'ambiente LIB. Se, ad esempio, tali librerie fossero all'interno di una directory di nome DOS del disco C, la variabile d'ambiente si dovrebbe preparare nel seguente modo

SET LIB=C:\DOS

Una volta creato il file QDOS.EXE (eseguibile), esso può essere eseguito richiamandolo dall'AUTOEXEC.BAT o direttamente.

APPENDICE A

I messaggi di errore

Appendice A

I messaggi di errore

In questa Appendice vengono elencati i messaggi di errore che il Quick Basic può fornire durante una delle varie fasi di realizzazione di un programma. Questo elenco, infatti, include gli **errori di chiamata** (**invocation errors**), gli **errori ed avvertimenti di compilazione** (**compile-time errors and warnings**), gli **errori e gli avvertimenti del linker** (**run-time errors**); vengono inoltre forniti gli **errori e gli avvertimenti del gestore di librerie LIB** (**LIB errors and warnings**).

Si userà la seguente codifica dei messaggi

* Invocation errors	**IE**
* Compile errors	**GE**
* Compile warnings	**CW**
* Run-time errors	**RE**
* Link time errors	**KE**
* Link time warnings	**KW**
* LIB errors	**BE**
* LIB warnings	**BW**

Advanced feature unavailable – Si sta tentando di usare una caratteristica di una versione superiore di QB o di MS-DOS (**CE, RE,** cod. **73**)

Argument-count mismatch – Si usano un numero errato di argomenti con una function o una sub (**CE**)

Array already dimensioned – Si usano più frasi DIM per un identico array statico, si usa una frase DIM dopo che un array è stato usato o dimensionato per default oppure si esegue una OPTION BASE dopo il dimensionamento di un array (**CE, RE**)

(name) : array-element size mismatch – Un array globale di tipo far è stato dimensionato, nel codice oggetto, più di una volta con gli elementi di tipo diverso (ad es., una volta come stringhe ed un'altra come interi); questo errore non si può presentare con il codice oggetto prodotto da compilatori Microsoft (**KE**, cod. **L2012**)

Array not defined – Si usa un array che non è stato definito (**CE**)

Array not dimensioned – Si usa un array che non è stato dimensionato (**CW**)

Array too big – L'array usato occupa più memoria di quella disponibile. Con gli arrays dinamici più grandi di 64 K si devono usare il metacomando $DYNAMIC (v. App. C) e l'opzione /ah nella linea di comando del QB (**CE**)

AS clause required – Se una variabile è dichiarata con la clausola AS e non con gli identificatori di tipo, deve essere usata con tale clausola anche in seguito nelle frasi COMMON, SHARED, DIM e REDIM (**CE**)

AS clause requie on first declaration – Se usando una variabile si include la clausola AS questa deve apparire nella dichiarazione della stessa variabile (**CE**)

AS missing – Manca la clausola AS in una frase OPEN… (**CE**)

Asterisk missing – Manca l'asterisco in una definizione di una stringa a lunghezza fissa (**CE**)

Attempts to access data outside segment bond – Nel codice oggetto, dei dati si estendono oltre la fine del segmento. Questo è un errore del traslatore (compilatore o assemblatore). Riferirsi all'assistenza software del produttore del traslatore (**KE**, cod. **L1103**)

Bad file mode – Si usano le istruzioni GET, PUT, FIELD con un file sequenziale o si tenta di usare un file sequenziale in modo diverso da come è stato aperto (si scrive in uno aperto in input o si legge da uno aperto in output o append); infine, può presentarsi questo errore quando si usa un file include registrato in formato binario e non in formato testo (**RE**, cod. **54**)

Bad file name – Si usano uno o più caratteri illegali nel nome di un file o una istruzione OPEN, SAVE, KILL o LOAD (**RE**, cod. **64**)

Bad file name or number – Si usa una istruzione che è collegata ad una frase OPEN in cui il nome o il numero di file non è corretto (**RE**, cod. **52**)

Bad record length – Una PUT o una GET tentano di agire con una variabile record la cui lunghezza è superiore a quella definita nella frase OPEN (**RE**, cod. **59**)

Bad record number – Non si usa un valore maggiore di zero come numero di record in una istruzione GET o PUT (**RE**, cod. **63**)

BASE missing – Non si è specificata la parola chiave BASE in una frase OPTION BASE (**CE**)

Binary source file – Si è tentato di compilare un file che non è in formato ASCII o formato Quick Basic; per il GW-BASIC e il BASICA usare l'opzione ,A nel comando SAVE (**CE**)

Block IF without END IF – Non si è trovato un END IF per un blocco IF precedente (**CE**)

Buffer size expected after /C: – Si deve specificare un numero per definire la grandezza del buffer di ricezione seriale dopo l'opzione /C: nella chiamata del compilatore bc (**IE**)

BYVAL allowed only with numeric arguments – La clausola BYVAL non ammette che parametri numerici (**CE**)

/C: buffer size too large – Si è usato un valore maggiore di 32767 per la definizione della grandezza del buffer seriale (**IE**)

(filename) : cannot access file – Il gestore LIB non può accedere al file specificato (**BE**, cod. **U2157**)

Cannot continue – Non è possibile più continuare l'esecuzione del programma; bisogna eseguirlo dall'inizio (**CE**)

(filename) : cannot create extract file – Non è possibile creare il file extract; il disco può essere pieno o il file extract già esistente può essere protetto da scrittura (**BE**, cod. **U1182**)

(filename) : cannot create listing – Non è possibile creare il file di cross- reference; il disco può essere pieno o il file di cross-reference già esistente può essere protetto da scrittura (**BE**, cod. **U2152**)

(filename) : cannot create new library – Non è possibile creare il file di libreria; il disco può essere pieno o il file di libreria già esistente può essere protetto da scrittura (**BE**, cod. **U1185**)

Cannot create temporary file – Non è possibile creare il file temporaneo; il disco può essere pieno (**KE**, cod. **L1084**)

Cannot find file (filename). Input path: – Non è possibile trovare il file indicato (generalmente una Quick Lib o un file di utility come LINK.EXE o LIB.EXE); si può specificare da tastiera il percorso adeguato o premere Ctrl-C per rinunciare (**IE**)

(filename) : cannot find file – Il linker non può trovare su disco e directory specificate il file indicato (**KW**, cod. **L4054**)

(filename) : cannot find library – Il linker non può trovare su disco e directory specificate il file di libreria indicato (**KW**, cod. **L4051**)

Cannot generate listing for BASIC binary source file – Non è possibile specificare l'opzione /a con il programma bc se il file sorgente è registrato in binario (**IE**)

Cannot nest response file – Non è possibile nidificare i response file (**KE**, cod. **L1021**)

Cannot open list file – Non è possibile aprire il file di list; la directory principale o il disco sono pieni (**KE**, cod. **L1080**)

Cannot open response file – Il response file non è stato trovato su disco (**BE**, cod. **U1183**)

(filename) : cannot open response file – Il linker non può aprire il file response specificato perché non esiste (**KE**, cod. **L1089**)

Cannot open run file – Non è possibile aprire il file .EXE in quanto il disco o la directory principale sono pieni (**KE**, cod. **L1083**)

Cannot open temporary file – Non è possibile aprire il file temporaneo in quanto il disco o la directory principale sono pieni (**KE**, cod. **L1085**)

Cannot open VM.TMP – Non è possibile aprire il file VM.TMP in quanto il disco o la directory principale sono pieni (**BE**, cod. **U1187**)

Cannot read from VM – Non è possibile leggere il file VM. Contattare la Microsoft (**BE**, cod. **U1189**)

Cannot rename old library – Il gestore LIB non può rinominare la vecchia libreria perché la vecchia libreria con estensione .BAK esiste ed è protetta da scrittura (**BE**, cod. **U1161**)

Cannot reopen library – La vecchia libreria non può essere riaperta dopo che è stata rinominata (**BE**, cod. **U1162**)

Cannot reopen list file – Non è possibile riaprire il file di list (**KE**, cod. **K1090**)

Cannot start with 'FN' – Non si possono usare le lettere FN per iniziare il nome di una variabile, di una sub o function, in quanto riservate (**CE**)

Cannot write to VM – Non è possibile scrivere il file VM. Contattare la Microsoft (**BE**, cod. **U1188**)

CASE ELSE expected – Il risultato di una espressione usata in una SELECT CASE non era stato previsto nella struttura e la clausola CASE ELSE non è stata specificata; nella versione 4.5 tale errore non si presenta e il programma continua dopo la END SELECT (**RE**, cod. **39**)

CASE without SELECT – È stata usata la parola chiave CASE senza avere impostato la struttura SELECT CASE in precedenza (**CE**)

(name) : code segment size exceeds 65500 – La lunghezza del segmento di codice è maggiore di 65500 bytes, il massimo consentito dal LINK con il processore Intel 80286 (**KW**, cod. **L4020**)

/CODEVIEW disable /DSALLOCATE – Le due opzioni non possono essere utilizzate contemporaneamente; la prima annulla la seconda (**KW**, cod. **L4015**)

/CODEVIEW disable /EXEPACK – Le due opzioni non possono essere utilizzate contemporaneamente; la prima annulla la seconda (**KW**, cod. **L4016**)

Colon expected after /C – Manca il simbolo : dopo l'opzione /C (**IE**)

Comma missing – Era richiesta una virgola che non è stata trovata (**CE**)

Comma or new line missing – Nel comando specificato il gestore LIB si aspettava una virgola o un ritorno di carrello che invece manca. È un errore di sintassi (**BE**, cod. **U1157**)

COMMON and DECLARE must precede executable statements – Esistono delle frasi COMMON e DECLARE in posizione seguente delle istruzioni eseguibili; le frasi COMMON e DECLARE devono precedere le istruzioni (**CE**)

Common area longer than 65536 bytes – Il codice oggetto presenta più di 65536 bytes occupati da variabili globali (**KE**, cod. **L1072**)

COMMON in Quick library too small – Esistono più variabili in un blocco COMMON del modulo corrente che non nel corrispondente blocco nella Quick library caricata (**CE**)

COMMON name illegal – Non è possibile usare il nome del blocco COMMON specificato in quanto riservato (**CE**)

Communication-buffer overflow – Il buffer di ricezione seriale si è riempito; settare la grandezza di tale buffer con l'opzione /C: del programma bc o con l'opzione RB nell'istruzione OPEN (**RE**, cod. **69**)

CONST / DIM SHARED follows SUB / FUNCTION – Esistono delle istruzioni CONST o DIM SHARED dopo le SUB o FUNCTION e ciò può far funzionare male tutto il programma (**CW**)

Control structure in IF…THEN…ELSE incomplete – È stata incontrata in una linea IF…THEN…ELSE una istruzione NEXT, END IF, END SELECT, LOOP, WEND non collegata ad istruzioni precedenti (**CE**)

(number) : CPARMAXALLOC : illegal value – Il numero specificato nell'opzione /CPARMAXALLOC non è all'interno del range 1-65535 (**KE**, cod. **L1009**)

Data-memory overflow – Esistono troppe costanti o arrays statici in un programma; usare gli arrays dinamici (**CE**)

Data record too large – In un modulo oggetto un record di tipo LEDATA contiene più di 1024 bytes di dati; è un errore del traslatore. Contattare la Microsoft (**KE**, cod. **L1057**)

DECLARE required – Una chiamata implicita (senza CALL) di una SUB o Function appare prima della dichiarazione della stessa (**CE**)

DEF FN not allowed in control statements – È stata dichiarata una fnuzione utente con la DEF FN all'interno di una struttura IF…END IF o SELECT CASE..END SELECT (**CE**)

DEF without END DEF – Manca la frase di chiusura di una funzione utente multi linea dichiarata in precedenza (**CE**)

DEFtype character specification illegal – È stato usato un tipo non corretto di dato in una istruzione DEFtype; sono permessi soltanto i tipi INT, LNG, SNG, DBL e STR (**CE**)

Device fault – È stato ritornato un codice di errore hardware da una periferica; è comune con l'uso della OPEN COM se la comunicazione non può avere luogo (**RE**, cod. **25**)

Device I/O error – È avvenuto un errore di ingresso/uscita durante una operazione con una periferica (**RE**, cod. **57**)

Device timeout – Non è stata mandata alcuna informazione da una periferica interrogata entro un determinato tempo (**RE**, cod. **24**)

Device unavailable – La periferica non esiste o non è collegata correttamente (**RE**, cod. **68**)

Disk full – Non esiste spazio sufficiente su disco per completare una operazione di scrittura dati (**RE**, cod. **61**)

Disk-media error – È stato rilevato un errore determinato da un danno fisico del disco (**RE**, cod. **72**)

Disk not ready – Il disco non è inserito correttamente nel drive su cui si tenta la scrittura o la lettura (**RE**, cod. **71**)

Division by zero – È stata tentata l'operazione di divisione per zero (**RE**, cod. **11**)

DO without LOOP – Non esiste l'istruzione LOOP terminante una struttura DO…LOOP (**CE**)

Duplicate definition – Si sta dimensionando con DIM un array già dimensionato oppure si sta definendo una variabile, costante, sub o function il cui nome è già usato (**CE**, **RE**, cod. **10**)

Duplicate label – Si sono usate delle etichette o dei numeri di linea uguali in due o più linee diverse del sorgente (**CE**)

Dynamic array element illegal – Non è consentito l'uso di elementi di arrays dinamici con la funzione VARPTR$ (**CE**)

Element not defined – L'elemento specificato della variabile utente in uso non è stato definito nella TYPE…END TYPE (**CE**)

ELSE without IF – Esiste una clausola ELSE senza una corrispondente IF (**CE**)

ELSEIF without IF – Esiste una clausola ELSEIF senza la corrispondente IF (**CE**)

END DEF without DEF – È stata usata una END DEF senza la corrispondente DEF (**CE**)

END IF without block IF – Manca l'inizio di una struttura IF a blocco (**CE**)

END SELECT without SELECT – È stata incontrata una END SELECT senza la corrispondente SELECT CASE iniziale (**CE**)

END SUB or END FUNCTION must be last line in window – In una SUB o in una FUNCTION non devono essere presenti istruzioni dopo la END SUB o END FUNCTION; usare la EXIT SUB o la EXIT FUNCTION per rientrare al codice chiamante prima del termine della SUB o FUNCTION (**CE**)

END SUB / FUNCTION without SUB / FUNCTION – Manca l'istruzione SUB (o FUNCTION) ed esiste una END SUB (o END FUNCTION) (**CE**)

END TYPE without TYPE – Una istruzione END TYPE è stata incontrata senza una istruzione TYPE corrispondente (**CE**)

Equal sign missing – È necessario un segno di eguale che invece manca (**CE**)

Error during QuickBASIC initialization – È un errore che si manifesta quando si tenta di caricare QB con un hardware che non lo può supportare o quando esiste poca memoria per caricarlo (**IE**)

Error in loading file (file) - Cannot find file – Esiste un errore nella posizione dei parametri della linea di invocazione del programma QB (**IE**)

Error in loading file (file) - Disk I/O error – Esiste un errore hardware che non consente di caricare il file indicato (**IE**)

Error in loading file (file) - DOS memory-arena error – L'area di memoria usata dal DOS è stata sovrascritta da routines Assembler o da istruzioni che fanno uso diretto della memoria (**IE**, **RE**)

Error in loading file (file) - Disk I/O error – Esiste un errore hardware che non consente di caricare il file indicato (**IE**)

Error in loading file (file) - Invalid format – Si sta tentando di caricare un file, una Quick Library o una LIB library il cui formato, per motivi diversi, non è compatibile con Quick Basic (**IE**)

Error in loading file (file) - Out of memory – È richiesta più memoria di quella disponibile dai dati e dai programmi (comprese le librerie) (**IE**, **RE**, cod. **7**)

Error writing to cross-reference file – È stato rilevato un errore durante la scrittura del file di cross-reference; il disco è probabilmente pieno (**BE**, cod. **U1163**)

Error writing to new library – È stato rilevato un errore durante la scrittura della nuova libreria;il disco è probabilmente pieno (**BE**, cod. **U1186**)

EXIT DO not within DO…LOOP – È stata incontrata una istruzione EXIT DO al di fuori di una struttura DO…LOOP (**CE**)

EXIT not within FOR…NEXT – È stata incontrata una istruzione EXIT FOR al di fuori di una struttura FOR…NEXT (**CE**)

Expected: (item) – Il compilatore si aspettava il carattere, la parola o il dato specificato; è un errore di sintassi (**CE**)

Expression too complex – L'espressione è troppo complessa oppure fa troppo uso di costanti alfanumeriche (**CE**)

Extra file name ignored – Si sono specificati troppi nomi di file nella linea di comando del programma QB; quelli in più sono ignorati (**IE**)

FIELD overflow – Con una istruzione FIELD si tenta di allocare più spazio in un record di quello dichiarato con la OPEN corrispondente (**RE**, cod. **50**)

FIELD statement active – Una GET o una PUT tentano di usare una variabile record in un file in cui i campi sono già stati definiti con una istruzione FIELD (**RE**, cod. **56**)

File already exist – Il file usato nell'istruzione NAME è quello usato correntemente (**RE**, cod. **58**)

File already open – Si è tentato di eseguire una istruzione KILL o una OPEN sequenziale in uscita quando il file indicato era già stato aperto (**RE**, cod. **55**)

File not found – Una qualsiasi istruzione che gestisce i file tenta di usarne uno che non esiste su disco (**RE**, cod. **53**)

File not suitable for /EXEPACK; relink without – Il linker ha rilevato che la grandezza del file compattato più quella della testata aggiunta supera quella del file non compattato; si consiglia di rieseguire il link senza l'opzione /EXEPACK (**KE**, cod. **L1114**)

File previosly loaded – Si sta tentando di caricare un file che è già in memoria (**CE**)

Fixed-length string illegal – Non è possibile usare una stringa di lunghezza fissa come parametro formale di una SUB o Function (**CE**)

Fixup overflow near (number) in frame seg (segname) target seg (segname) target offset (number) – Questo errore può essere causato dalle seguenti condizioni : un gruppo è più grande di 64 K; il codice oggetto contiene un salto vicino tra diversi segmenti; il nome di un dato nel codice oggetto entra in conflitto con un dato contenuto in una libreria; una dichiarazione EXTRN nel sorgente non è posta al di fuori della pseudoistruzione SEGMENT (**KE**, cod. **L2002**)

Fixup(s) without data – È stato trovato nel codice oggetto un record di tipo fixup senza il relativo record di dati posto prima; è un errore del traslatore. Contattare la Microsoft (**KE**, cod. **L2001**)

Fixup type unsupported – Esiste nel codice oggetto un tipo fixup che non è supportato dal linker; è caratteristico di una incompatibilità con il traslatore. Contattare la Microsoft (**KE**, cod. **L2005**)

FOR index variable already in use – La variabile di controllo di due cicli FOR nidificati, è la stessa (**CE**)

FOR index variable illegal – È consentito solamente l'uso di variabili numeriche semplici come indice di cicli FOR (**CE**)

FOR without NEXT – Non esiste una istruzione NEXT corrispondente al FOR indicato (**CE**)

Formal parameter specification illegal – Esiste un errore nella specifica degli argomenti di una SUB o Function (**CE**)

Formal parameters not unique – La frase di dichiarazione di una SUB o Function contiene dei parametri con nome duplicato (**CE**)

Free: not allocated – È un errore interno del gestore LIB. Contattare la Microsoft (**BE**, cod. **U1175**)

Function already defined – È stata ridefinita una Function (**CE**)

Function name illegal – È stata usata qualche parola chiave riservata nel nome di una Function (**CE**)

Function not defined – È stata usata una Function non definita o dichiarata (**CE**)

GOSUB missing – Manca l'istruzione GOSUB in una istruzione ON event (**CE**)

GOTO missing – Manca l'istruzione GOTO in una istruzione ON ERROR (**CE**)

GOTO or GOSUB expected – Manca una istruzione GOTO o GOSUB che invece deve esistere (**CE**)

Identifier cannot end with %, &, !, #, or $ – I suddetti suffissi non sono consentiti dall'istruzione che ha generato l'errore (**CE**)

Identifier cannot include period – Non è possibile usare il punto all'interno di una variabile; questo è usato come separatore tra la variabile e gli elementi di un dato definito dall'utente (**CE**)

Identifier expected – Il compilatore si aspettava una variabile dove è stato posto invece un numero o una parola riservata del Quick Basic (**CE**)

Identifier too long – Il nome dell'identificatore non può essere più lungo di 40 caratteri (**CE**)

Illegal function call – È stato usato un valore che eccede i limiti consentiti dalla funzione chiamata (**RE**, cod. **5**)

Illegal in direct mode – L'istruzione che ha generato l'errore, non può essere usata nella Immediate Window ma solo in un programma (**CE**)

Illegal in procedure or DEF FN – L'istruzione non è consentita all'interno di una procedura o in una DEF FN..END DEF (**CE**)

Illegal number – Il formato del numero costante specificato non corrisponde a quello consentito dal Quick Basic (**CE**)

Illegal outside of SUB, FUNCTION or DEF FN – L'uso dell'istruzione specificata non è consentito a livello di modulo (**CE**)

Illegal outside of SUB / FUNCTION – L'uso dell'istruzione specificata è consentito solo all'interno di una SUB o Function (**CE**)

Illegal outside of TYPE block – La frase che ha generato l'errore è consentita solo all'interno di un blocco TYPE..END TYPE (**CE**)

Illegal type character in numeric constant – Una costante numerica contiene, al termine, un carattere di dichiarazione di tipo illegale (**CE**)

$INCLUDE - file access error – Il file indicato nel metacomando $INCLUDE (v. App. C) non può essere raggiunto (**CE**)

Include file too large – Il file da includere è troppo grande; deve essere suddiviso in parti più piccole (**CE**)

Input file not found – Il file sorgente specificato nel comando QB non può essere trovato (**IE**)

INPUT missing – Il compilatore si aspettava la parola chiave INPUT che invece manca (**CE**)

Input past end of file – Una istruzione di lettura da file sequenziale tenta di leggere da un file vuoto o da uno già letto (**RE**, cod. **62**)

Input runtime module path: – Il modulo runtime BRUN40.EXE (o BRUN45.EXE per la versione 4.5) non può essere trovato; si può specificare il path in cui può essere trovato. Questo run-time error non può essere gestito con la ON ERROR (**RE**)

Insufficient memory – Il gestore LIB richiede più memoria di quella disponibile per lavorare (**BE**, cod. **U1171**)

Insufficient memory, extended dictionary not created – Il gestore LIB non può creare il dizionario esteso perché non ha a disposizione la memoria necessaria (**BW**, cod. **U4157**)

Integer between 1 and 32767 required – Viene richiesto un valore intero nei limiti specificati (**CE**)

Internal error – Un malfunzionamento interno del compilatore genera questo errore. In questo caso bisogna contattare la Microsoft compilando un modulo per Product Assistant Request (**RE**, cod. **51**)

Internal error, extended dictionary non created – Il gestore LIB non può creare il dizionario esteso perché è avvenuto un errore interno. Contattare la Microsoft (**BW**, cod. **U4158**)

Internal error near xxxx – Come per l'errore precedente, con la differenza che questo si manifesta all'indirizzo xxxx che deve essere indicato alla Microsoft (**CE**)

Internal failure – Si è verificato un errore interno del gestore LIB. Contattare la Microsoft (**BE**, cod. **U1173**)

(option) : interrupt number exceeds 255 – È stato specificato un numero maggiore di 255 come parametro nell'opzione /OVERLAYINTERRUPT (**KE**, cod. **L1007**)

Invalid character – È stato incontrato un carattere non valido all'interno di un file sorgente (**CE**)

Invalid constant – L'espressione usata per assegnare un valore ad una costante non è corretta; essa fa probabilmente uso di operatori non consentiti in questo caso (**CE**)

Invalid DECLARE for BASIC procedure – Non devono essere usate le clausole ALIAS, CDECL e BYVAL nella dichiarazione di una procedura BASIC; esse sono, infatti, riservate alle procedure scritte in altri linguaggi (**CE**)

(filename) : invalid format (hexnumber); file ignored – Il codice di riconoscimento del tipo di file oggetto non è valido; è possibile usare solo file oggetto in formato Microsoft library, Intel library, Microsoft object e Xenix archive (**BE**, cod. **U2159**)

(libraryname) : invalid library header – La libreria in input ha un formato non corretto o è danneggiata (**BE**, cod. **U1200**)

(libraryname) : invalid library header; file ignored – La libreria in input ha un formato non corretto e viene ignorata (**BE**, cod. **U2158**)

(option) : invalid numeric value – Un valore non corretto è stato immesso per una opzione del linker (**KE**, cod. **L1004**)

Invalid object module – Uno o più file oggetto non sono validi. Se sono stati generati con traslatori della Microsoft, contattare quest'ultima (**KE**, cod. **L1101**)

(name) : invalid object module near (location) – Il modulo oggetto specificato da name, non è valido per il gestore LIB (**BE**, cod. **U1203**)

Label not defined – È stata usata una etichetta che non esiste in una istruzione di salto (**CE**)

Left parenthesis missing – Il compilatore si aspettava una parentesi aperta (**CE**)

LIDATA record too large – Un record di tipo LIDATA è più grande di 512 byte; è un errore del traslatore (**KE**, cod. **L2013**)

Line invalid. Start again – Un errore nel nome dei file è stato fatto durante l'invocazione del programma BC (**IE**)

Line number or label missing – In una istruzione di salto manca il riferimento ad una etichetta o numero di linea (**CE**)

Line too long – Una linea in un testo sorgente è più lunga dei 255 caratteri consentiti (**CE**)

Load-high disables EXEPACK – Le due opzioni non possono essere usate contemporaneamente. La /HIGH disabilita la /EXEPACK (**KW**, cod. **L4012**)

LOBYTE-type fixup overflow – Un fixup LOBYTE genera un overflow di indirizzamento; contattare la Microsoft (**KE**, cod. **L2004**)

LOOP without DO – L’istruzione DO in una struttura DO…LOOP non è stata specificata (**CE**)

Lower bound exceeds upper bound – In una frase DIM il valore minimo degli elementi di una dimensione è superiore a quello massimo (**CE**)

Mark : not allocated – È un errore interno del gestore LIB. Contattare la Microsoft (**BE**, cod. **U1174**)

Math overflow – Il risultato dell’espressione è troppo grande per rappresentarlo con un numero in Quick Basic (**CE**)

$Metacommand error – È stato usato un metacomando (v. App. C) in maniera non corretta (**CE**)

Minus sign missing – Manca il segno ‘meno’ (-) (**CE**)

Missing Event Trapping (/W) or Checking Between Statements (/V) option – Dato che il programma contiene una istruzione ON event, è necessario l’uso di uno degli switch suddetti durante la compilazione (**CE**)

Missing On Error (/E) option – È stata usata l’istruzione ON ERROR e non è stata specificata l’opzione /E nella compilazione (**CE**)

Missing Resume Next (/X) option – È stata usata l’istruzione RESUME e manca l’opzione /X nella linea di comando per la compilazione (**CE**)

Module level code too large – Il codice a livello di modulo è troppo grande; usare delle SUB e delle Function per snellirlo (**CE**)

(modulename) : module not in library – Il modulo che si sta tentando di sostituire non è presente in libreria; ne viene aggiunto uno nuovo (**BW**, cod. **U4155**)

(modulename) : module not in library; ignored – Il modulo specificato non è stato trovato nella libreria di input (**BE**, cod. **U2155**)

(modulename) : module redefinition ignored – Il modulo specificato è già presente in memoria al momento dell’inserimento o sono presenti due copie dello stesso in libreria (**BW**, cod. **U4150**)

More than 239 overlay segments; extra put in root – Sono stati specificati più di 239 segmenti di overlay; quelli extra sono posti nella porzione residente del programma (**KW**, cod. **L4034**)

Must be first statement on line – Nella struttura a blocco le istruzioni IF..ELSE..ELSEIF..ENDIF devono essere le prime nella linea (fatta eccezione per eventuali etichette) (**CE**)

Name of subprogram illegal – Il nome di un sottoprogramma contiene parole chiave riservate o è usato due volte (**CE**)

Name of output file is (name) – Il nome del file di output dato dal compilatore non è quello indicato dallo stesso a causa di un cattivo uso dell'opzione /QUICKLIB; il nome esatto viene specificato da questo messaggio d'avvertimento (**KE**, cod. **4045**)

(name) : NEAR / HUGE conflict – Gli attributi NEAR e HUGE entrano in conflitto nella definizione di variabili common; è un errore del traslatore (**KE**, cod. **L2011**)

Nested function definition – La definizione di una Function è all'interno di un'altra Function o all'interno di una struttura IF..THEN..ELSE (**CE**)

Nested left parenthesis – La specifica degli overlays sulla riga di comando non è esatta; controllare le parentesi aperte (**KE**, cod. **L1025**)

Nested right parenthesis – La specifica degli overlays sulla riga di comando non è esatta; controllare le parentesi chiuse (**KE**, cod. **L1024**)

NEXT without FOR – Manca l'istruzione FOR per la NEXT specificata (**CE**, **RE**, cod. **1**)

No line number in (module name) at address (segment:offset) – Non è possibile trovare un numero di linea in cui si è verificato un errore perché mai specificato. È un errore che non può essere gestito con una ON ERROR (**RE**)

No main module. Choose Set Main Module from the Run menu to select one – Non è possibile compilare ed eseguire un programma in cui manchi il modulo principale (**CE**)

No more virtual memory – È un errore interno del gestore di librerie; contattare la Microsoft (**BE**, cod. **U1172**)

No object modules specified – Non è stato specificato alcun modulo oggetto al linker (**KE**, cod. **L1020**)

No RESUME – Non è presente alcuna routine di gestione degli errori prima della fine del modulo pur essendo stata usata la ON ERROR (**RE**, cod. **19**)

No stack segment – Non è stato specificato alcun segmento di tipo STACK; il compilatore Quick Basic provvede alla definizione di tale segmento; l'errore, quando si manifesta con file oggetto assembler, può essere ignorato se il segmento non è stato creato volutamente (**KW**, cod. **L4021**)

(filename) : not valid library – Il file indicato non è una libreria valida o è danneggiato (**KE**, cod. **1104**)

Not watchable – Non è possibile accedere alle variabili specificate nella Watch window (**RE**)

Numeric array illegal – Non è possibile usare degli arrays numerici come argomenti con la funzione VARPTR$ (**CE**)

Object not found – Uno dei file oggetto indicati al linker non esiste (**KE**, cod. **L1093**)

Only simple variables allowed – È solamente consentito l'uso di variabili semplici con le istruzioni che generano questo errore (**CE**)

Operation requires disk – Si tenta di usare una istruzione di scrittura o lettura da disco con una periferica che non lo consente (**CE**)

(option) : option name ambiguous – Quanto specificato per una opzione non consente al linker di riconoscerla univocamente; inserire più lettere che la chiariscano (**KE**, cod. **L1001**)

Option unknown – Una opzione sconosciuta è stata fornita al gestore LIB (**BE**, cod. **U1154**)

Option unknown: (option) – È stata usata una opzione sconosciuta con il programma BC (**IE**)

Out of DATA – È stata eseguita un'istruzione READ quando non esistevano altre frasi DATA da leggere (**RE**, cod. **4**)

Out of data space – È terminato lo spazio per i dati (**CE**, **RE**, cod. **7**)

Out of memory – Non c'è più memoria a disposizione del programma e dei dati (**IE**, **CE**, **RE**, cod. **7**)

Out of memory for symbol table – È terminata la memoria a disposizione della tabella dei simboli; definire meno simboli pubblici possibile (**KE**, cod. **L1053**)

Out of paper – La stampante non ha più carta o è disattivata (**RE**, cod. **24**)

Out of space for list file – Il file di list non può essere scritto su disco; probabilmente il disco stesso è pieno (**KE**, cod. **L1088**)

Out of space for run file – Il file eseguibile non può essere scritto su disco; probabilmente il disco stesso è pieno (**KE**, cod. **L1081**)

Out of stack space – Una Function ricorsiva ha richiesto troppo stack; modificare la grandezza dello stack con CLEAR. È un errore che non è possibile gestire con la ON ERROR (**RE**)

Out of string space – Le stringhe usate eccedono lo spazio disponibile (**RE**, cod. **14**)

(libraryname) : output library specification ignored – È stata specificata una libreria di output nella linea di comando insieme ad una nuova libreria (**BW**, cod. **U4156**)

Overflow – Il risultato dell'espressione è troppo grande per essere trattato con il tipo di dato specificato (**RE**, cod. **6**)

Overflow in numeric constant – La costante numerica specificata è troppo grande per il suo tipo (**CE**)

Page size too small – L'ampiezza della pagina della libreria di input è troppo piccola; ciò indica una libreria non compatibile (**BE**, cod. **U1150**)

(number) : page size too small; ignored – Il valore indicato nell'opzione /PAGESIZE è minore di 16; l'indicazione è ignorata e viene assunto il valore di default (**BW**, cod. **U4153**)

Parameter type mismatch – Nella chiamata di una Function non corrisponde il tipo di parametri usati nella DECLARE e nella FUNCTION (**CE**)

Path not found – Il path specificato nelle istruzioni OPEN, MKDIR, CHDIR e RMDIR non è corretto (**RE**, cod. **76**)

Path/File access error – Il DOS non è stato capace di accedere al path/file specificato nelle istruzioni OPEN, MKDIR, CHDIR e RMDIR (**RE**, cod. **75**)

Permission denied – È stato tentato un accesso ad un disco protetto in scrittura o ad un file bloccato con LOCK (**RE**, cod. **70**)

Procedure already defined in Quick Library – Una procedura contenuta nella Quick Library caricata ha lo stesso nome di una procedura attualmente in memoria (**CE**)

Procedure too large – La procedura indicata è troppo grande (**CE**)

Program memory overflow – Si sta tentando di compilare un programma il cui segmento di codice è maggiore di 64 K; bisogna dividerlo in più moduli o usare l'istruzione CHAIN (**CE**)

/QUICKLIB, /EXEPACK incompatible – Le due opzioni specificate non possono essere usate contemporaneamente (**KE**, cod. **L1003**)

Quick library support module missing – Non è stato specificato o non è stato trovato il supporto per la creazione delle quick libraries; per il Quick Basic questo è il file BQLB40.LIB (**KE**, cod. **L2043**)

/QUICKLIB, overlays incompatible – Non è possibile indicare l'opzione /QUICKLIB e, contemporaneamente, degli overlays (**KE**, cod. **L1115**)

Read error on standard input – È avvenuto un errore durante la lettura di dati da console o da un file rediretto in input (**IE**)

Record / string assignment required – È necessaria l'istruzione LSET per assegnare la variabile record o la stringa specificata (**CE**)

Redo from start – Si sono forniti ad una istruzione INPUT dei dati in numero diverso da quelli richiesti; reinserirli (**RE**)

Relocation table overflow – Sono stati specificati più di 32768 salti, chiamate o puntatori lunghi; usare, dove possibile, riferimenti vicini (**KE**, cod. **L1043**)

Rename across the disk – Si è tentato di rinominare un file specificando drive diversi nel vecchio e nuovo nome dello stesso (**RE**, cod. **74**)

Requested segment limit too high – Non esiste memoria sufficiente per allocare la tabella di descrizione dei segmenti specificati; usare l'opzione /SEGMENT per diminuire il numero di segmenti o modificare il file sorgente (**KE**, cod. **L1054**)

Requires DOS 2.10 or later – Si tenta di eseguire il Quick BASIC con un DOS precedente a quello specificato (**IE**, **RE**)

Response line too long – Una linea in un response file è più lunga di 127 caratteri (**KE**, cod. **L1022**)

Resume without error – È stata incontrata un'istruzione RESUME senza che accadesse alcun errore (**RE**, cod. **20**)

RETURN without GOSUB – È stata incontrata un'istruzione RETURN senza la corrispondente GOSUB (**RE**, cod. **3**)

Right parenthesis missing – Il compilatore si aspettava una parentesi chiusa (**RE**)

Scratch file missing – È un errore interno del linker. Contattare la Microsoft (**KE**, cod. **L1086**)

SEG or BYVAL not allowed in CALLS – Le clausole SEG e BYVAL non sono consentite nell'istruzione CALLS ma solo nella CALL (**CE**)

(name) : segment declared in more than one group – Un segmento è stato dichiarato come appartenente a due o più gruppi diversi (**KW**, cod. **L4031**)

(option) : segment limit set too high – È stato specificato un numero di segmenti più alto di 3072 con l'opzione /SEGMENT (**KE**, cod. **K1008**)

Segment size exceeds 64 K – Un singolo segmento contiene più di 64 K tra codice e dati; ricompilare usando un modello di memoria più capace (**KE**, cod. **L1070**)

Segment _TEXT larger than 65520 bytes – Il segmento _TEXT è più grande di 65520 bytes; questo errore può essere generato quando si usa un compilatore che usa il segmento _TEXT e viene specificata l'opzione /DOSSEG (**KE**, cod. **L1071**)

SELECT without END SELECT – Non esiste la corrispondente istruzione END SELECT di una SELECT CASE (**CE**)

Semicolon missing – Manca un carattere ; dove invece richiesto (**CE**)

Separator illegal – È stato usato un delimitatore vietato nelle istruzioni PRINT USING o WRITE; sono permessi il ; e la , (**CE**)

Simple or array variable expected – Il compilatore si aspettava una variabile o un elemento di un array (**CE**)

Skipping forward to END TYPE statement – Un errore in una struttura TYPE..END TYPE fa sì che il compilatore ignori quanto in questa contenuto (**CE**)

Stack plus data exceed 64 K – Lo spazio occupato dai dati più quello per lo stack è superiore a 64 K (**KE**, cod. **L2041**)

(option) : stack size exceeds 65535 bytes – Il valore indicato con l'opzione /STACKSIZE è superiore al massimo visualizzato (**KE**, cod. **L1006**)

Statement cannot occur within $INCLUDE file – Le istruzioni SUB..END SUB e FUNCTION..END FUNCTION non possono apparire in un file include (**CE**)

Statement cannot precede SUB/FUNCTION definition – È possibile far precedere le definizioni di SUB e Function solo dalle istruzioni DEFtype e REM (**CE**)

Statement ignored – Si sta compilando con il programma BC un file che contiene le istruzioni TRON e TROFF senza usare l'opzione /d; tali istruzioni sono ignorate (**CW**)

Statement illegal in TYPE block – È stata usata una istruzione non consentita all'interno di un blocco TYPE..END TYPE (**CE**)

Statement unrecognizable – È stata incontrata una istruzione non riconoscibile dal compilatore (**CE**)

Statements/labels illegal between SELECT CASE and CASE – Sono state usate istruzioni non consentite o etichette tra la SELECT CASE e la prima istruzione CASE (**CE**)

STOP in module (name) at address (segment:offset) – È stata incontrata una istruzione STOP nel programma (**RE**)

String assignment required – È necessario usare l'istruzione RSET nell'assegnazione della stringa (**CE**)

String constant required for ALIAS – È necessaria una costante nel parametro usato con la clausola ALIAS nell'istruzione DECLARE (**CE**)

String expression required – È necessario un argomento di tipo stringa (**CE**)

String formula too complex – Una espressione stringa è troppo complessa per essere eseguita senza doverla spezzare in due o più parti oppure una istruzione INPUT richiede più di 15 variabili stringa (**RE**, cod. **16**)

String space corrupt – Questo errore è generato quando, durante la garbage collection delle stringhe, viene eliminata una stringa il cui descrittore è stato danneggiato da routines

assembler non corrette o da istruzioni che, in maniera errata, accedono direttamente in memoria. Non è un errore recuperabile (**RE**)

String variabile required – L'istruzione richiede una variabile di tipo stringa (**CE**)

SUB or FUNCTION missing – Non esiste la corrispondente SUB o Function di una DECLARE esistente (**CE**)

SUB/FUNCTION without END SUB/FUNCTION – L'istruzione di fine di una SUB o Function non è stata incontrata (**CE**)

Subprogram error – È un errore generato quando una Function o un sottoprogramma è già stato definito o è nidificato in maniera non corretta (**CE**)

Subprogram not defined – Un sottoprogramma è chiamato ma non definito (**CE**)

Subprograms not allowed in control statements – Le istruzioni SUB e FUNCTION non sono permesse all'interno di costrutti IF e SELECT CASE (**CE**)

Subscript out of range – È stato usato un elemento di un array che non era stato definito o un elemento con indice che non rientra nei limiti della definizione dello stesso (**RE**, cod. **9**)

Subscript syntax illegal – La definizione delle dimensioni di un array è sintatticamente errata (**CE**)

(name) : symbol already defined – È stato trovato un simbolo pubblico definito più di una volta (**KE**, cod. **L2024**)

(name) : symbol defined more than once – È stato trovato un simbolo pubblico definito più di una volta (**KE**, cod. **L2025**)

Symbol multiply defined, use /NOE – È stato trovato un simbolo pubblico definito più di una volta; è possibile che questo sia stato ridefinito in una libreria; usare l'opzione /NOEXTDICTIONARY (**KE**, cod. **L2044**)

(symbol) : symbol redefined in module (modulename), redefinition ignored – Il simbolo indicato è stato definito in più di un modulo; tutte le ridefinizioni sono ignorate (**BW**, cod. **U4151**)

Syntax error – Il comando per l'uso del gestore LIB non rispetta la sua sintassi (**BE**, cod. **U1156**)

Syntax error – È un errore generato da una istruzione scritta in maniera errata nel file sorgente o da un cattivo uso di READ e DATA (**CE**, **RE**, cod. **2**)

Syntax error : illegal file specification – Un operatore (+ - *) nella linea di comando non è stato specificato insieme ad un file espresso correttamente (**BE**, cod. **1151**)

Syntax error : illegal input – Il comando per l'uso del gestore LIB non rispetta la sua sintassi (**BE**, cod. **U1155**)

Syntax error in numeric constant – Una costante numerica non è scritta in maniera corretta (**CE**)

Syntax error : option name missing – È stata specificata, nella linea di comando, una barra (/) ma non è stata indicata l'opzione (**BE**, cod. **U1152**)

Syntax error : option value missing – Non è stato specificato un valore per l'opzione indicata nella linea di comando (**BE**, cod. **U1153**)

Terminated by user – È stato premuto il Ctrl-Break (Ctrl-C) durante l'operazione di link (**KE**, cod. **L1023**)

Terminator missing – Manca il carattere di ritorno carrello nell'ultima linea del response file (**BE**, cod. **U1158**)

THEN missing – Manca l'istruzione THEN in un costrutto IF (**CE**)

TO missing – Manca la parola chiave TO dove richiesta (**CE**)

Too many arguments in function call – Sono stati usati più argomenti di quelli consentiti nella chiamata di una Function o di una SUB (**CE**)

Too many dimension – Sono state definite più dimensioni di quelle consentite nell'istruzione per la dichiarazione di un array (**CE**)

Too many external symbols in one module – In un singolo modulo sono definiti più di 1023 simboli esterni (**KE**, cod. **1046**)

Too many file – Sono stati usati più di 5 livelli di nidificazione dei file include oppure è stato generato questo errore da una istruzione OPEN o SAVE che non può svolgere le proprie funzioni (**CE**, **RE**, cod. **67**)

Too many groups – Sono stati definiti più di 20 gruppi oltre DGROUP (**KE**, cod. **L1051**)

Too many groups in one module – Sono stati definiti più di 21 gruppi in un singolo modulo (**KE**, cod. **L1050**)

Too many group, segment, and class names in one module – Sono stati definiti troppi gruppi, segmenti o classi in un singolo modulo (**KE**, cod. **L1047**)

Too many labels – Il numero di etichette seguenti le istruzioni ON..GOTO e ON..GOSUB è eccessivo (**CE**)

Too many libraries – È stato richiesto il link dopo avere specificato più di 32 librerie diverse (**KE**, cod. **L1052**)

Too many named COMMON blocks – È stato superato il numero massimo di 126 blocchi COMMON permessi (**CE**)

Too many overlays – Sono stati definiti più di 63 overlays (**KE**, cod. **L1056**)

Too many public symbols for sorting – Sono stati definiti troppi dimboli pubblici per poterli ordinare e presentare secondo le specifiche dell'opzione /MAP; i simboli sono presentati senza essere ordinati (**KW**, cod. **L2050**)

Too many segments – Sono stati usati più di 128 segmenti o più segmenti di quanto specificato con l'opzione /SEGMENT (**KE**, cod. **L1049**)

Too many segments in one module – Un singolo modulo presenta più di 255 segmenti (**KE**, cod. **L1048**)

Too many symbols – Nel file di libreria esistono più di 4609 simboli (**BE**, cod. **U1170**)

Too many TYPDEF records – Esistono più di 255 records di tipo TYPDEF (**KE**, cod. **L1045**)

Too many TYPE definitions – È stato superato il numero massimo di 240 tipi di dati definiti dall'utente permessi (**CE**)

Too many variables for INPUT – È stato superato il numero massimo di 60 variabili usabili dall'istruzione INPUT (**CE**)

Too many variables for LINE INPUT – È consentita solo una variabile con l'istruzione LINE INPUT (**CE**)

Type mismatch – È stato usato un tipo di dato errato (**CE**, **RE**, cod. **13**)

TYPE missing – Manca la parola chiave TYPE in una istruzione END TYPE (**CE**)

Type more than 65535 bytes – Un tipo definito dall'utente non può essere più lungo di 64 K (**CE**)

Type not defined – Il tipo utente non è stato definito (**CE**)

TYPE statement improperly nested – Non sono consentite definizioni di dati utente in procedure (**CE**)

TYPE without END TYPE – Manca l'istruzione END TYPE in una struttura TYPE..END TYPE (**CE**)

Typed variable not allowed in expression – Le variabili di tipo utente non sono consentite nell'espressione che ha generato l'errore (**CE**)

Unexpected end-of-file – Un file di libreria è terminato in maniera inaspettata; non è di tipo corretto o è danneggiato (**KE**, cod. **L1102**)

Unexpected end-of-file in TYPE declaration – È terminato il file sorgente ma non la struttura TYPE..END TYPE (**CE**)

Unexpected end-of-file on command input – È stato fornito un carattere di end-of-file in risposta ad una domanda in input del gestore di librerie (**BE**, cod. **U1184**)

Unexpected end-of-file on library – È stato rilevato una carattere di end-of-file inaspettato nella libreria (**KE**, cod. **L1091**)

Unexpected end-of-file on scratch file – È stato rilevato un carattere di end-of-file inaspettato nel file temporaneo; rimuovere il disco in cui è contenuto (**KE**, cod. **L1087**)

Unmatched left parenthesis – Manca una parentesi aperta nella linea di comando nella specifica degli overlays (**KE**, cod. **L1027**)

Unmatched right parenthesis – Manca una parentesi chiusa nella linea di comando nella specifica degli overlays (**KE**, cod. **L1026**)

Unprintable error – Un messaggio di errore non è disponibile per questo errore; questo errore viene generato dall'istruzione ERROR quando il codice relativo non corrisponde ad alcun errore stampabile (**RE**)

(option) : unrecognized option name – Non è stata riconosciuta una opzione indicata nella linea di comando (**KE**, cod. **L1002**)

Unrecognized switch error: "QU" – Si sta creando una Quick Lib o un file .EXE con una versione di linker sbagliata; si deve usare la versione Microsoft Overlay (**CE**)

Unresolved COMDEF; internal error – È un errore interno del linker; contattare la Microsoft (**KE**, cod. **L1113**)

Unresolved externals – Uno o più simboli dichiarati come esterni in uno o più moduli non sono stati dichiarati pubblici in nessuna libreria o modulo (**KE**, cod. **L2029**)

Valid options: [RUN] file /AH /B /C:buf /G /H /L [lib] /MBF /CMD string – È stato richiamato il programma QB con una opzione illegale (**IE**)

Variable-length string required – È necessario usare una stringa a lunghezza variabile (**CE**)

Variable name not unique – Si sta tentando di usare una variabile di tipo utente diverso da come era stata dichiarata in precedenza (**CE**)

Variable required – È necessario usare una variabile (**CE**, **RE**, cod. **40**)

VM.TMP : illegal file name; ignored – È stato indicato il file VM.TMP come file oggetto; il file viene ignorato (**KW**, cod. **L4053**)

WEND without WHILE – Ad una istruzione WEND non corrisponde una WHILE (**CE**)

WHILE without WEND – Ad una istruzione WHILE non corrisponde una WEND (**CE**)

Write to extract file failed – Il file extract non può essere scritto; probabilmente il disco è pieno (**BE**, cod. **U1180**)

Write to library file failed – Il file di libreria non può essere scritto; probabilmente il disco è pieno (**BE**, cod. **U1181**)

Wrong number of dimension – Un riferimento ad un array contiene un numero errato di dimensioni rispetto a quelle preparate (**CE**)

APPENDICE B

I file include della libreria BPLUS

Appendice B

I file include della libreria BPLUS

In questa appendice viene riportato il contenuto di ogni file include appartenente alla libreria BPLUS. Questi file, come descritto nel cap. 7, contengono tutte le dichiarazioni di funzioni e procedure e quelle dei dati globali usati dalle stesse, divise per gruppi.

```
'
' Libreria BPLUS - File include BPLUS.INC
'        Antonio Giuliana (c) 1991
'
DEFINT A-Z
' $DYNAMIC
DECLARE SUB ABEEP ()
DECLARE FUNCTION ADDZERO2STR$ (X!, CF)
DECLARE SUB BEEPNO ()
DECLARE SUB BEEPYES ()
DECLARE SUB CAPSOFF ()
DECLARE SUB CAPSON ()
DECLARE FUNCTION CENTREWIN (OpWin, Astr$, Col, Ret, Row)
DECLARE FUNCTION CINPUT (Px, Py, Lun, OldS$, NewS$, Col, Typ, Prompt)
DECLARE SUB CLEARBOX (Xt, Yt, Xb, Yb, Col)
DECLARE FUNCTION CLEARWINDOW (OpWin, Col)
DECLARE SUB CLOSEPOPMENU (Menu, Handle$)
DECLARE FUNCTION CLOSEWINDOW (OpWin)
DECLARE FUNCTION CMPDATE (D1$, D2$)
DECLARE FUNCTION COLORWINDOW (OpWin, Col)
DECLARE FUNCTION CONVATTR$ (CATTR)
DECLARE FUNCTION CURRDISK$ ()
DECLARE FUNCTION CURRPATH$ (DISK$)
DECLARE FUNCTION CURSORWINDOW% (OpWin, Row, Col)
DECLARE FUNCTION DISKFREE& (DISK$)
DECLARE FUNCTION DISKTYPE$ (DISK$)
DECLARE FUNCTION DOSVER$ ()
DECLARE FUNCTION DWEEK (DT$)
DECLARE FUNCTION ETA (DN$, DO$)
DECLARE SUB EXPEXT (TEXP, FEXP, FEXT)
DECLARE FUNCTION FILEDATE$ (D$)
DECLARE FUNCTION FILEEXIST (FILE$)
DECLARE FUNCTION FILETIME$ (T$)
DECLARE FUNCTION GETDCC$ ()
DECLARE SUB GETDTA ()
DECLARE FUNCTION GETFILEATTR$ (FILE$)
DECLARE FUNCTION GETKEYMENU (Menu)
DECLARE FUNCTION GETLABEL$ (DISK$)
DECLARE FUNCTION GETMOUSEBUTTON ()
DECLARE SUB GETMOUSEPOS (POSX, POSY)
```

```
DECLARE FUNCTION GETVIDEOSEG ()
DECLARE SUB INVERSE (OpWin, Inv, Largh, Col)
DECLARE FUNCTION ISDIGIT (Ch)
DECLARE FUNCTION MCHOICE (Xt, Yt, Xb, Yb, Title$, Bottom$, Col, ArData$(), SelPos)
DECLARE FUNCTION MOUSE (OPER, COOX, COOY, BUTTON)
DECLARE FUNCTION MOUSECHECK ()
DECLARE SUB MOUSEOFF ()
DECLARE SUB MOUSEON ()
DECLARE SUB MVDATA CDECL (BYVAL Sse, BYVAL Sof, BYVAL Dse, BYVAL Dof, BYVAL Sz)
DECLARE FUNCTION OPENPOPMENU (Menu, Handle$)
DECLARE FUNCTION OPENWINDOW (Xt, Yt, Xb, Yb, Col, Crn)
DECLARE SUB PRINTSCREEN ()
DECLARE FUNCTION PRINTWINDOW (OpWin, Astr$, Col, Ret)
DECLARE SUB PRNRESET (PR AS INTEGER)
DECLARE FUNCTION PRNSTATUS$ (PR AS INTEGER)
DECLARE FUNCTION PUTKEY (Ch$)
DECLARE SUB PUTSTRING (S$)
DECLARE FUNCTION READDIR$ (FILE$, ONLYDIRS, NF, ND, FS AS LONG)
DECLARE SUB RESTBOX (Xt, Yt, Xb, Yb, Handle$)
DECLARE FUNCTION SAVEBOX$ (Xt, Yt, Xb, Yb)
DECLARE SUB SCROLLDNBOX (Xt, Yt, Xb, Yb, NL, Col)
DECLARE SUB SCROLLUPBOX (Xt, Yt, Xb, Yb, NL, Col)
DECLARE FUNCTION SELECTOPZMENU! (MaxMenus, StMenu)
DECLARE SUB SETDISK (DISK$)
DECLARE SUB SETDTA (DSEG AS INTEGER, DOFF AS INTEGER)
DECLARE SUB SETFILEATTR (FILE$, ATTR$)
DECLARE SUB SETMOUSEPOS (POSX, POSY)
DECLARE SUB SETPOPMENU (MaxMenus)
DECLARE SUB STATUSROW (AreaA$, AreaB$, AreaC$)
DECLARE FUNCTION TODAY$ ()

'
' Libreria BPLUS - File include TYPES.INC
'       Antonio Giuliana (c) 1991
'
DEFINT A-Z
TYPE RegType
        AX      AS INTEGER
        BX      AS INTEGER
        CX      AS INTEGER
        DX      AS INTEGER
        BP      AS INTEGER
        SI      AS INTEGER
        DI      AS INTEGER
        FLAGS   AS INTEGER
END TYPE
TYPE RegTypeX
        AX      AS INTEGER
```

```
        BX      AS INTEGER
        CX      AS INTEGER
        DX      AS INTEGER
        BP      AS INTEGER
        SI      AS INTEGER
        DI      AS INTEGER
        FLAGS   AS INTEGER
        DS      AS INTEGER
        ES      AS INTEGER
END TYPE
TYPE Wind
        Used AS INTEGER
        InRow AS INTEGER
        InCol AS INTEGER
        Rows AS INTEGER
        Cols AS INTEGER
        CurRow AS INTEGER
        CurCol AS INTEGER
        Crn AS INTEGER
        PrecAreaSeg AS INTEGER
END TYPE
TYPE DirArea
        DOSRESERVED AS STRING * 21
        FILEATT AS STRING * 1
        FILETIME AS STRING * 2
        FILEDATE AS STRING * 2
        FILEIZE AS STRING * 4
        FILENAME AS STRING * 13
        FILLDIR AS STRING * 85
END TYPE

'
'  Libreria BPLUS - File include COMMON.INC
'       Antonio Giuliana (c) 1991
'
COMMON SHARED /Windows/  Windows() AS Wind
COMMON SHARED /PopMenu/  PopMenus() AS STRING, AltKeysMenu(), BeepYesNo
COMMON SHARED /Dos/  DtaSeg, DtaOff, Reg AS RegType, Regx AS RegTypeX
COMMON SHARED /Dir/  Dir AS DirArea, Names() AS STRING, Size() AS LONG
COMMON SHARED /Dir/  Typ() AS STRING, FDate() AS STRING, FTime() AS STRING
COMMON SHARED /DateTime/  Giorno() AS STRING, Mese() AS STRING

'
'  Libreria BPLUS - File include DOS.INC
'       Antonio Giuliana (c) 1991
'
DECLARE SUB INTERRUPT (Intnum AS INTEGER, Inreg AS RegType, Outreg AS RegType)
DECLARE SUB INTERRUPTX (Intnum AS INTEGER, Inreg AS RegTypeX, Outreg AS RegTypeX)
```

```
‘
‘  Libreria BPLUS - File include POPMENU.INC
‘          Antonio Giuliana (c) 1991
‘
DEFINT A-Z
CONST MAXM = 10
DUM& = SETMEM (-MAXM * 500&)
DIM SHARED PopMenus(1 TO MAXM) AS STRING
DIM SHARED AltKeysMenu (1 TO MAXM)
BeepYesNo = 0

‘
‘  Libreria BPLUS - File include WIN.INC
‘          Antonio Giuliana (c) 1991
‘
DEFINT A-Z
CONST MAXW = 10
DUM& = SETMEM (-MAXM * 4000&)
DIM SHARED Windows (1 TO MAXW) AS Wind

‘
‘  Libreria BPLUS - File include DIR.INC
‘          Antonio Giuliana (c) 1991
‘
DEFINT A-Z
Entry = 512
DIM SHARED Names(1 TO Entry) AS STRING, Size(1 TO Entry) AS LONG
DIM SHARED Typ(1 TO Entry) AS STRING
DIM SHARED FDate(1 TO Entry) A STRING, FTime(1 TO Entry) AS STRING

‘
‘  Libreria BPLUS - File include DATETIME.INC
‘          Antonio Giuliana (c) 1991
‘
DIM SHARED Giorno(0 TO 6) AS STRING
Giorno$(0) = “Domenica”
Giorno$(1) = “Lunedi”
Giorno$(2) = “Martedi”
Giorno$(3) = “Mercoledi”
Giorno$(4) = “Giovedi”
Giorno$(5) = “Venerdi”
Giorno$(6) = “Sabato”
DIM SHARED Mese(1 TO 12) AS STRING
Mese$(1) = “Gennaio”
Mese$(2) = “Febbraio”
Mese$(3) =  “Marzo”
Mese$(4) =  “Aprile”
```

```
Mese$(5) = “Maggio”
Mese$(6) =  “Giugno”
Mese$(7) =  “Luglio”
Mese$(8) =  “Agosto”
Mese$(9) =  “Settembre”
Mese$(10) =  “Ottobre”
Mese$(11) =  “Novembre”
Mese$(12) =  “Dicembre”
```

APPENDICE C

I metacomandi

Appendice C

I metacomandi

I metacomandi del Quick Basic sono delle direttive che si impartiscono al compilatore per trattare i programmi dell'utente in maniera diversa, a seconda delle situazioni che si presentano.

Queste direttive risolvono i seguenti due problemi che si possono presentare durante la realizzazione dei programmi

1. Inclusione di file di testata nel testo sorgente
2. Uso di array statici e/o dinamici

La sintassi dei metacomandi prevede che essi vengano preceduti dal simbolo $ e che siano posti all'interno di una riga di commento, come nei seguenti esempi

REM $metacomando

' $metacomando

Il metacomando usato per l'inclusione dei file di testata è il seguente

$INCLUDE: 'nome file incluso'

Esso va usato nel punto del sorgente in cui si vuole includere il file di testata tenendo presenti le seguenti regole

- il file incluso deve essere di tipo testo
- nel file incluso non devono esistere SUB o FUNCTION
- è possibile includere fino a 5 file nidificandoli

La compilazione del sorgente riprende esattamente dove era stata interrotta per iniziare quella del file incluso.

I metacomandi relativi all'uso degli array statici o dinamici sono i seguenti

$STATIC
e
$DYNAMIC

Il primo fa si che gli arrays definiti nelle righe che lo seguono, vengano definiti come statici ed allocati durante la compilazione; per questi arrays l'istruzione ERASE azzera tutti gli elementi mentre la REDIM non opera.

Il secondo permette di definire tutti gli arrays inizializzati in seguito, come dinamici e cioè allocati al momento dell'esecuzione della relativa istruzione DIM. Essi possono occupare tutta la memoria convenzionale a disposizione del sistema e su loro il comando ERASE agisce in

maniera da eliminarli completamente dalla memoria restituendola al sistema per altri usi; l’istruzione REDIM inoltre, è utile per fare in modo di cambiare il numero degli elementi (ma non le dimensioni) di un array di questo tipo.

Notare che, nella sintassi dei metacomandi, è obbligatorio almeno uno spazio tra l’istruzione REM ed il metacomando stesso e tra il metacomando ed un suo eventuale parametro. Non è consentito invece alcun carattere tra il simbolo $ ed il metacomando e tra quest’ultimo ed il simbolo : (v. $INCLUDE:).

Indice

QUICK BASIC

Tecniche di programmazione avanzata

CAPITOLO 1

L'INSTALLAZIONE E L'AVVIO

1.1 CARATTERISTICHE GENERALI

1.1.1 Caratteristiche del sistema 9
1.1.2 Installazione su floppy disk 9
1.1.3 Installazione su hard disk 11
1.1.4 Se si dispone di un coprocessore aritmetico 11
1.1.5 Se si dispone di un mouse 12
1.1.6 Caratteristiche del Quick Basic v. 4.0 12
1.1.7 Differenze con le precedenti versioni 15
1.1.8 Importanti caratteristiche aggiunte al linguaggio 15

1.2 IL COMANDO QB

1.2.1 Il comando QB ed i relativi parametri 16
1.2.2 La configurazione dello schermo di Quick Basic 17
1.2.3 Il file QB.INI 18

CAPITOLO 2

L'AMBIENTE INTEGRATO

2.1 L'AMBIENTE DI SVILUPPO DEI PROGRAMMI

2.1.1 Il comando QB ed i relativi parametri 21
2.1.2 Scelta delle opzioni dai menu 22
2.1.3 Uso delle finestre 22
2.1.4 La dialog box 23

2.2 IL MENU FILE

2.2.1 I diversi tipi di file 23
2.2.2 Gestione dei programmi 24
2.2.3 Le SUBs e le FUNCTIONs 26
2.2.4 I moduli ed i file .MAK 27
2.2.5 Gli altri tipi di file 29
2.2.6 La stampa dei file 29
2.2.7 L'uscita da Quick Basic 30

2.3 IL MENU EDIT

2.3.1 Immissione e controllo del testo 30
2.3.2 Selezione del testo 32
2.3.3 La memoria di transito per l'edit (Clipboard) 32
2.3.4 Cancellazione, spostamento e copia del testo 32
2.3.5 Il comando Undo 33
2.3.6 Immissione di simboli speciali 34
2.3.7 Le opzioni New SUB… e New FUNCTION… 34
2.3.8 Tabella di riepilogo dei tasti di edit 35

2.4 IL MENU SEARCH
2.4.1 Ricerca del testo 36
2.4.2 Ricerca e modifica del testo 37
2.4.3 Uso dei marcatori di testo 38
2.4.4 Tabella di riepilogo dei tasti di ricerca 38

2.5 IL MENU VIEW
2.5.1 Lo schermo di output 38
2.5.2 La gestione delle SUBs 39
2.5.3 La divisione dello schermo in ambiente Quick Basic 39
2.5.4 Gestione dei file include 40

CAPITOLO 3
REALIZZARE UN PROGRAMMA

3.1 CREAZIONE DEI PROGRAMMI
3.1.1 Uso della Immediate Window 43
3.1.2 Immissione delle istruzioni di un programma 43
3.1.3 Creazione di un Main Module 44
3.1.4 Creazione di un programma multi-modulo 44
3.1.5 Il file .MAK 46
3.1.6 Cambio del Main Module 47
3.1.7 Gestione dei moduli in un programma multi-modulo 47

3.2 IL MENU RUN
3.2.1 Esecuzione dei programmi 48
3.2.2 La variabile di sistema COMMAND$ 48
3.2.3 Creazione di file .EXE 49
3.2.4 Creazione delle Quick Libraries 50

3.3 IL DEBUGGING

3.3.1 Esecuzione di istruzioni singole 50
3.3.2 Esecuzione di parti di programma e procedure singole 50
3.3.3 Uso delle opzioni Trace ed History 50
3.3.4 I Breakpoints e l'opzione Set Next Statement 51
3.3.5 La Watch Window ed i Watchpoints 51
3.3.6 Il menu Calls 52

3.4 IL MENU DI AIUTO
3.4.1 Generalità sull'aiuto in linea 53
3.4.2 Le pagine di aiuto generale 53
3.4.3 Aiuto context-sensitive 54
3.4.4 Differenze con l'aiuto della versione 4.5 di QB 54

CAPITOLO 4
LA COMPILAZIONE

4.1 LA COMPILAZIONE
4.1.1 Il programma BC 56
4.1.2 Il programma LINK 58

4.2 GESTIONE DELLE LIBRERIE
4.2.1 Il concetto di libreria 62
4.2.2 Le librerie .LIB 62
4.2.3 Le librerie .QLB 63
4.2.4 Il gestore di librerie Microsoft LIB 63
4.2.5 Esempio di realizzazione ed uso di una libreria 65

CAPITOLO 5
LANGUAGE REFERENCE

5.1 IL LINGUAGGIO DI PROGRAMMAZIONE
5.1.1 Le istruzioni e i comandi di Quick Basic 70
5.1.2 Le funzioni di Quick Basic 227

CAPITOLO 6
PROGRAMMARE CON IL QB

6.1 LE BASI PER PROGRAMMARE IN QUICK BASIC

6.1.1 Tipi di dati in Quick Basic 285
6.1.2 Tipi di dati definiti dall'utente 288
6.1.3 Costanti e variabili 291
6.1.4 Variabili globali, locali, statiche ed automatiche 295
6.1.5 Gli arrays 299
6.1.6 Il passaggio degli argomenti alle Subs e Functions 302
6.1.7 Le espressioni e gli operatori 304

6.2 LE STRUTTURE DI CONTROLLO
6.2.1 Uso delle strutture decisionali 309
6.2.2 Uso delle strutture cicliche 313

6.3 IL TRATTAMENTO DELLE STRINGHE
6.3.1 Tipi di stringhe 319
6.3.2 Unione e comparazione di stringhe 319
6.3.3 Trattamento di parti di stringhe 323

6.4 USARE I FILE
6.4.1 La gestione dei file 326
6.4.2 Condivisione di file in multiutenza 342

6.5 LA GRAFICA
6.5.1 Cenni sulle potenzialità grafiche 343

CAPITOLO 7
IL QB E GLI ALTRI LINGUAGGI

7.1 QBASIC, C ED ASSEMBLER
7.1.1 Generalità sull'interfacciamento ad altri linguaggi 349
7.1.2 Interfacciamento al Macroassembler v. 5.0 Microsoft 350
7.1.3 Esempi di interfacciamento al MASM v. 6.0 356
7.1.4 Interfacciamento al MSC Compiler v. 6.0 373
7.1.5 Esempi di interfacciamento al MSC Compiler v. 6.0 375

7.2 BPLUS LIBRARY
7.2.1 La libreria BPLUS 379
7.2.2 Routines in Quick Basic commentate 382
7.2.3 Uso della libreria BPLUS : programma di esempio 423

APPENDICI

A. I messaggi di errore 442
B. I file Include della libreria BPLUS 466
C. I metacomandi 473